2007

An algebraic hash function based on SL2

Andrew Richard Regenscheid
Iowa State University

An algebraic hash function based on SL_2

by

Andrew Richard Regenscheid

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Clifford Bergman, Major Professor
Sung Song
Siu-Hung Ng

Iowa State University

Ames, Iowa

2007

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Cryptographic hash functions are fundamental building blocks of many computer security systems and protocols, primarily being used to ensure data integrity. Recent attacks against modern hash functions have questioned the suitability of standard hash function construction principles. In this paper we consider a hash function construction based multiplication in the group of 2×2 matrices over a finite field proposed by Zémor and Tillich [48, 42, 43]. We also look at how the algebraic properties of hash functions following this design can be exploited in attacks. Finally, we consider variations to the approach of Zémor and Tillich that offer some resistance to those attacks.

CHAPTER 1. INTRODUCTION

Cryptographic hash functions are one of the fundamental building blocks in many computer security systems. They are among the most versatile cryptographic primitives finding applications in a variety of areas. Cryptographic hash functions are typically used to provide data integrity, attempting to prove that a piece of data is not modified accidently or intentionally. However, they often find themselves used in a other seemingly unrelated areas, such as pseudorandom number generation and password storage. Each security application involving hash functions needs a different set of properties fulfilled by the hash function to achieve security. Creating general hash functions which meet the needs of all or most of these applications is difficult.

Unlike the other primary class of cryptographic primitives, encryption functions, most researchers believe there is still much to learn about proper hash function construction. Largely to build off the knowledge base of encryption functions, most hash functions today are constructed using an iterative design centered on block ciphers. There is a large body of research on attacks against such hash functions. Until recently, however, these attacks did not pose a credible security risk to hash functions in use. That quickly changed in 2004 and 2005 when serious attacks were developed against the two most commonly used hash functions, MD5 and SHA-1.

Efforts are currently underway to replace those hash functions. Some have suggested tweaking the current design of hash functions to be resistant to current attacks. Others have suggested creating new constructions for hash function design that are still related to the design of encryption functions. A third option is to create a hash function using algebraic structures. The mathematical structure of these hash functions often leads to some attacks, but that same

structure can sometimes prove limited claims about the hash function's security.

In [48, 42, 43], Zémor and Tillich propose two algebraic hash functions which are based on computations in the special linear group of 2×2 matrices over a finite field $\mathbb{F}_p$ or $\mathbb{F}_{2^n}$. The hash function construction is related to the Cayley graph of the matrix group with a generating pair. Colliding messages can be thought of as two distinct paths through this graph that both start at the identity matrix at end at some arbitrary group element. Certainly the high degree of algebraic structure in this hashing scheme opens the door for attacks that exploit this structure.

This paper examines the hashing schemes proposed by Zémor and Tillich, as well as the major attacks that have been developed against them. Chapter 2 introduces the concept of hash functions and summarizes classical hash function design. In Chapter 3 we describe the original hashing scheme over $SL_2(\mathbb{F}_p)$ developed and analyzed by Zémor and Tillich, as well as the flaws in the scheme. Chapter 4 describes the improved hash function which operates over $SL_2(\mathbb{F}_{2^n})$ and the two major attacks that have been proposed. Finally, Chapter 5 discusses variations of the basic hash function design.

CHAPTER 2. CRYPTOGRAPHIC HASH FUNCTIONS

Cryptographic hash functions are algorithms designed to provide a level of assurance for data integrity. These functions generally take as input an arbitrarily long message and output a short digital fingerprint of the input data. This fingerprint is often referred to as a *message digest*.

To understand the purpose of hash functions, consider the following motivating example. In computer forensics it is necessary to provide assurances that digital evidence has not been modified purposefully or accidently during the examination process. Prior to working with digital evidence such as a hard drive, forensic analysts often obtain a message digest of the original evidence. After the investigation is completed, any individual can compute the message digest for the current state of the evidence and compare it to the stored message digest of the original evidence. If the digests match, the individual can have some level of assurance that the evidence was not modified.

2.1 Definition

The study of hash functions can be separated into into two cases: *keyed and unkeyed hash functions*. While this paper will focus on unkeyed hash functions, we begin by providing the definition from [40] for the more general case of keyed hash functions, also known as hash families.

Definition 2.1.1. A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ where the following conditions are satisfied

1. $\mathcal{X}$ is a set of possible messages

2. $\mathcal{Y}$ is a finite set of possible message digests

3. $\mathcal{K}$, the keyspace, is a finite set of possible keys

4. For each $k \in \mathcal{K}$ there is a function $h_k \in \mathcal{H}$ where h_k is a function from $\mathcal{X}$ to $\mathcal{Y}$.

In particular, *unkeyed hash functions* can be defined as a hash family with only one possible key. The functions in $\mathcal{H}$ are known as hash functions. A hash function can be defined as follows.

Definition 2.1.2. A hash function is a function h with the following two properties:

1. Compression: h maps an input from a domain $\mathcal{X}$ of large or infinite order to a codomain $\mathcal{Y}$ of smaller order.

2. Ease of Computation: For an arbitrary $x \in \mathcal{X}$, $h(x)$ is easy to compute.

As an unkeyed hash function is a hash family with only one hash function, the terms unkeyed hash function and hash function are used interchangeably.

2.2 Properties of Secure Hash Functions

While definitions 2.1.1 and 2.1.2 provide the baseline requirements for a hash function, a function meeting only those requirements would not be suitable for security applications, where they are commonly referred to as cryptographic hash functions. Cryptographic hash functions are hash functions which meet one or more of the following properties.

1. Preimage Resistance:

   Given $y \in \mathcal{Y}$, it is infeasible to find a message $x \in \mathcal{X}$ such that $h(x) = y$.

2. Second Preimage Resistance:

   Given a message $x \in \mathcal{X}$, it is infeasible to find a second message $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

3. Collision Resistance:

   It must be infeasible to find two message x, x' such that $x \neq x'$ but $h(x) = h(x')$.

Certainly, collision resistance implies second preimage resistance. For this reason, collision resistance is often referred to as strong collision resistance, while second preimage resistance is weak collision resistance. However, there need not be any relationship between preimage resistance and either type of collision resistance. Consider the following examples from [28, Ch. 9].

Example 2.2.1. A hash function may be preimage resistant, but not collision resistant. Consider $h(x) = x^2 \pmod{n}$ where $n = pq$ and p, q are large primes. Since computing a square root is equivalent to factoring n in this case, this function is preimage resistant. However, since $h(x) = h(-x)$ this function is not collision resistant or second primage resistant.

Alternatively, a hash function may be collision resistant, but not preimage resistant.

Example 2.2.2. Let g be a collision resistant hash function mapping to bitstrings of length $n-1$. Then define a hash function h by

$$h(x) = \begin{cases} 1 \mid\mid x & \text{if } x \text{ has bitlength } n-1 \\ 0 \mid\mid g(x) & \text{otherwise.} \end{cases}$$

Finding a collision in h is equivalent to finding a collision in g. Thus, h is a collision resistant hash function. Nonetheless, it is a trivial matter to find a preimage of any message digest beginning with a 1-bit.

In addition, there are several other properties which are desirable, but not always necessary. The specific properties needed are dependent upon the how the hash function will be used. Examples of such properties held by commonly used hash function include the following.

1. *Avalanche Property* Every input bit should affect every output bit.

2. *Near-Collision Resistance* It should be infeasible to find any two inputs $x, x' \in \mathcal{X}$ such that $h(x)$ and $h(x')$ differ by a small amount.

3. *Partial Preimage Resistance* Given $y \in \mathcal{Y}$ it should be infeasible to find even part of a message $x \in \mathcal{X}$ such that $h(x) = y$.

2.3 Traditional Hash Function Construction

Modern hash function construction is traditionally done using an iterative approach. Most commonly used hash functions today use an approach known as MD Construction, named after Ralph Merkle and Ivan Damgård who independently developed the design in [29] and [11].

The key idea behind the Merkle-Damgård construction is that a hash function accepting nearly infinite length bitstrings can be constructed using a compression function that maps bitstrings of fixed length to bitstrings of shorter length.

Definition 2.3.1. A *compression function* is a function that maps elements from $\{0,1\}^{m+t}$ to elements of $\{0,1\}^m$ where $m, t \in \mathbb{N}$.

The construction works as follows. Let f be a compression function from $\{0,1\}^{n+t} \rightarrow \{0,1\}^n$. Take an input message m and break it into substrings of length t, so $m = x_1 x_2 \dots x_l$, padding the last block with 0-bits as needed. An additional block, x_{l+1} holds the length of m.

The above steps are known as preprocessing the message. Next we iterate on the message blocks using our compression function. Define a fixed constant with bit-length n and denote it by H_0. Iteratively construct the following sequence

$$H_i = f(H_{i-1}||x_i)$$

from $1 \leq i \leq l+1$.

The final term in this sequence, H_{l+1}, can optionally undergo postprocessing. The result is the n-bit message digest for m. This hash function construction is summarized in Figure 2.1.

The inclusion of an additional block composed of the message length is known as *MD-strengthening*. This step prevents any padded message from being the tail end of any other padded message. By adding this step, Merkle and Damgård were able to prove in [29] and [11] that if the underlying compression function f is collision resistant, then the constructed iterated hash function is also collision resistant.

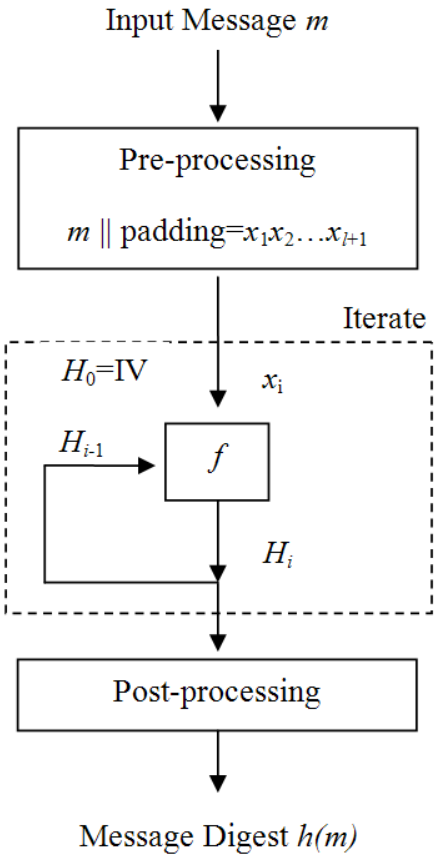

Figure 2.1 MD Construction

2.4 Compression Function Construction

Compression functions for use in iterated hash functions are typically constructed using algorithms resembling cryptographic block ciphers. In theory, widely accepted block ciphers such as the Advanced Encryption Standard [32] could be used in the construction, although typically new algorithms are designed for hash function construction for computational efficiency reasons.

Most current hash functions use the MD construction with a compression function based on the Davies-Meyer design. In such a design, the compression function in each iteration of the hash function is defined by

$$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$$

where $E_{x_i}(m)$ denotes encrypting m with algorithm E using key x_i. Thus, if E is an encryption function with a 192-bit key on 128-bit blocks, the resulting compression function maps bitstrings of length 320 to bitstrings of length 128.

The next basic approach essentially reverses the roles of the message block x_i and the result of the previous round H_{i-1}. Instead, the key fed to the block cipher is H_{i-1}. If the length of H_{i-1} is not equal to the key length of E then a fixed transformation function g is first applied to H_{i-1}. This approach is known as Matyas-Meyer-Oseas design and can be defined by

$$H_i = E_{g(H_{i-1})}(x_i) \oplus x_i.$$

A variation of the above design was developed by Shoji Miyaguchi and Bart Preneel, which modifies the algorithm above by exclusive-or-ing the result of each round with H_{i-1}.

2.5 Generic Attacks on Iterated Hash Functions

A common complaint against hash functions built on the Merkle-Damgård construction is that there are a variety of generic attacks that are effective against most hash functions of that design. These attacks all make use of the iterative structure of the design.

Iterative hash functions typically use the output of the last round of the compression function as the message digest, forgoing any final transformation. Because of this fact, nearly all MD-based hash functions suffer from the *length extension* property, which allows the attacker to calculate the hash value of an appended message without knowing the original message [16].

Consider an iterative hash function which is based on the MD construction that does not include a message length block. Denote this hash function by H and let f be its compression function. For any message with blocks $m_1, m_2, \ldots, m_i$, the output of the hash function is equal to $f(H(m_1 m_2 \ldots m_{i-1}), m_i)$. From this fact, we can see that if an attacker knows $H(m)$ for some message m the attacker can compute $H(m||m') = H(H(m)||m')$, for any message m'. This attack assumes m, if necessary, is padded to have a length which is a multiple of the block length of H.

MD-strengthening does not alleviate this attack if the attacker knows the length of the original message m. By implementing MD-strengthening, a hash function creates a final block

which encode the length of the hashed message. Thus, to perform this attack on a hash function that uses MD-strengthening the first block of the message m' must be chosen to be the binary representation of the length of m.

This same principle allows a class of attacks known as *second collision attacks* [27]. Let H be an iterated hash function without MD-strengthening and suppose an attacker finds two messages M, N such that $H(M) = H(N)$. The attacker can construct additional collisions because $H(M||S) = H(N||S)$. If H uses MD-strengthening the same result holds if the length of M and N are equal and if S includes, as the first message block, the encoded length of M.

More recently, Antoine Joux developed an attack which allows an attacker to find 2^k messages which hash to the same value using, on average, $k2^{n/2}$ hash operations [21]. Typically one would expect such multicollisions to be far more difficult to obtain.

To describe the attack, let H be an iterated hash function with compression function f. Find two message blocks that collide in the compression function from the initialization vector, i.e. $f(IV, B_0) = f(IV, B'_0) = h_0$. Next, find two additional blocks which collide with h_0. That is, find B_1, B'_1 such that $f(h_0, B_1) = f(h_0, B'_1) = h_1$. Continue this process up to finding blocks B_k, B'_k such that $f(h_{k-1}, B_k) = f(h_{k-1}, B'_k) = h_k$. Note that by the birthday paradox, if f outputs n bits, then finding a collision, on average, should take $2^{n/2}$ operations.

These blocks, B_i, B'_i, can be used to obtain 2^k collisions. Because of how they've been constructed, all messages of the form $b_1 b_2 \cdots b_k$ where $b_i \in \{B_i, B'_i\}$ hash to the same value. Joux highlights potential applications of this attack in [21].

Some iterated hash functions are also vulnerable to second-preimage attacks. That is, finding a second message which hashes to the same value as a given message. A brute force search on an n-bit hash function should take roughly 2^n operations to find such a second preimage. However work by Dean [12], and recently improved by Kelsey and Schneier [23], show that preimages can be found much more quickly. Their work is based on the notion of expandable messages. Expandable messages are messages of different lengths whose hash values, when calculated without a message length block are equal. The attacks in [12, 23] work against all hash functions built on the Merkle-Damgård construction.

2.6 Modern Hash Functions and Security

Currently, one of the most widely used hash functions is MD5 [35]. MD5 was developed by Ronald Rivest in 1991, and its widespread use comes from the fact that it was accepted as an Internet standard. As such, it finds use in validating X.509 certificates and Secure Socket Layers, the key element of secure websites today.

MD5 is an iterated hash function following the MD construction with a message digest consisting of 128 bits. It uses a tailor-made compression function consisting of 64 iterations of various bit operations and roughly follows the design principals of Davies-Meyer compression function construction.

The security of MD5 has been severely challenged in recent years by an increase in computational power and attacks. Because the size of the message digest is 128 bits, a brute force search would find a collision, on average, after 2^{64} attempts, barely within reach using current technology. Recently a serious vulnerability in the algorithm was discovered by Wang, Feng, Lai and Yu. In [46] they show how collisions can be found very quickly, reportedly only one hour on an IBM p690 cluster. The attack is described in detail in [20].

Merely being able to find collisions in a hash function does not necessarily mean that attacks against systems using the hash function are practical. In many cases, collision attacks have limited practical effect on security applications. However, in 2005, Wang discovered an attack involving a hash collision of two X.509 certificates [39]. This attack has serious security ramifications for Internet communications.

Attacks against MD5 have been expected for over a decade. Until recently, the most widely trusted hash function was SHA-1. SHA-1 was designed by the National Security Agency and accepted as a Federal Information Processing standard in 1995 [31]. The general design of SHA-1 is very similar to that of MD5. However, it uses a larger message digest, 160 bits, to limit the practicality of finding a collision using a brute force search.

In 2005 the same research group responsible for the attack on MD5 discovered a similar attack against SHA-1 [47]. The attack allows an attacker to find collisions after 2^{69} attempts- far less than the 2^{80} attempts expected in a brute force search. The research group has re-

cently announced an improved algorithm which would find collisions after 2^{63} hash operations. Consequently, the cryptographic community is moving away from SHA-1, and a competition to replace the SHA-1 federal standard is in the early stages of development.

While serious, the attacks against SHA-1 do not pose an immediate threat to practical security. In fact, despite these attacks being accepted as valid, no one has yet discovered a collision using the attacks on the SHA-1 algorithm.

2.7 Mathematical Hash Functions

Traditional hash function construction is based on performing several rounds of complex bit operations in a sequence that is difficult to reverse. Unfortunately, as shown by recent attacks against MD5 and SHA-1, there is little guaranteed security. Some cryptographers have long attempted to develop a hash function with a more trustworthy level of security by using algebraic structures.

One accepted, but little used, mathematical hash function is the Modular Arithmetic Secure Hash algorithm (MASH-1) [28]. The MASH algorithm roughly follows MD-construction by iterating over fixed length substrings of an arbitrary length message, but it uses a compression function based on modular arithmetic. The rounds of the compression function can be defined by

$$H_i = ((x_i \oplus H_{i-1}) \vee A)^2 \pmod{N} \oplus H_{i-1}$$

where $\vee$ denotes a bitwise and, N is a product of two primes, A is fixed constant, and x_i represents padded and expanded message blocks. While there is no proof of equivalence, the purpose of this structure is to make the security of the hash function related to finding roots modulo N.

Other approaches using algebraic structures are able to equate the difficultly of finding a collision to a known hard problem. One of the earliest was proposed by Damgård in [10]. Finding collisions in his scheme is equivalent to factoring an RSA modulus $n = pq$, where p, q are primes. Other schemes exist with security relying on the discrete logarithm problem

[18, 7, 3]. Beyond factoring and discrete logarithms, knapsack and lattice problems are other common sources for difficult problems [34].

Nearly all provably secure hash functions have two major drawbacks. First, the proven security properties are strictly limited to either collision resistance or preimage resistance, and do not prove that the hash function in question is suitable in practical applications. Also, the hash functions are particularly slow, which effectively prevents them from being used.

CHAPTER 3. A HASH FUNCTION BASED ON $SL_2(\mathbb{F}_p)$

3.1 Hash Function Construction

In 1991, Gilles Zémor proposed a hash function in [48] based on Cayley graphs. His original scheme was based on the Cayley graph of the group $SL_2(\mathbb{F}_p)$. First consider the following statement from [48].

Theorem 3.1.1. *The group $SL_2(\mathbb{F}_p)$ is generated by the following pairs of elements:*

1. $A_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $B_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$
2. $A_2 = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$, $B_2 = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$
3. $A_3 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$, $B_3 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$

The hash function proposed by Zémor is defined as follows. Let A, B be a pair of generators of $SL_2(\mathbb{F}_p)$ and let $m = b_0 b_1 \dots b_n$ be a binary string. Then $H(m) = \pi(b_0) \cdot \pi(b_1) \cdots \pi(b_n)$ where

$$\pi(b) = \begin{cases} A & \text{for } b = 0 \\ B & \text{for } b = 1 \end{cases}$$

This hash function is strongly related to the directed Cayley graph associated with $SL_2(\mathbb{F}_p)$ and generators A, B, denoted by $\mathcal{G}$. A message m can be considered as a directed path in $\mathcal{G}$ from the identity matrix to the value $H(m)$. In this context we can use graph-theoretic approaches to obtain proofs of security of this hash function.

3.2 Properties of the Hash Function Construction

Recall that the hash function construction presented above is directly associated with the directed Cayley graph $\mathcal{G}(G, S)$, where G is a group generated by the elements of the set S.

Definition 3.2.1. The directed girth of a graph $\mathcal{G}$ the largest integer τ such that given any two vertices v and w, any pair of distinct directed paths joining v to w will be such that one of those paths has length τ or more.

Considering the associated Cayley graph $\mathcal{G}(G, S)$, Definition 3.2.1 immediately leads to the following property of the hash function.

Proposition 3.2.2. *If we replace k consecutive elements of the product*

$$x = x_1 x_2 \cdots x_i x_{i+1} \cdots x_{i+k} x_{i+k-1} \cdots x_t$$

where $x_i \in S$, with h consecutive elements $y_{i+1}, \ldots, y_{i+h} \in S$ such that

$$x = x_1 x_2 \cdots x_i y_{i+1} \cdots y_{i+h} x_{i+k-1} \cdots x_t$$

then $\max(k, h) \geq \tau$.

That is, in order to obtain collisions from local modifications of text, such modifications must be made over at least τ bits, where τ is the girth of the Cayley graph associated with the hash function. In other words, suppose we were given a message and we wanted to change only a small portion of that message yet still retain the same hash value. For example, we may want to change a message like "I agree to pay \$100" to "I agree to pay \$500". According to Proposition 3.2.2, it would be impossible to change a message like this without changing the hash value. This provides a limited, but provable, measure of a hash function's collision resistance.

In practice, this property of the hash function would rarely provide protection against attacks. In many cases, collisions involving two very different messages can pose serious security threats. Also, the above property provides no assurance when its possible to change two small, distant, sections of text, as would be the case if "I agree to pay \$100" was changed to "I

hereby agree to pay \$500". That being said, there may be applications where this property would be useful. In some cases, hash functions are applied to messages that have very specific formats and fixed lengths. If one changes too much of such a message it may no longer follow the format and could thus be flagged as an invalid message. Under such circumstances, if the girth of the graph is large enough, this may effectively make it impossible to create a properly formatted colliding message.

For example, we can determine the girth of the Cayley graph associated with generators A_1 and B_1. To do this, note the following lemmas.

Lemma 3.2.3. *Let* $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ *and* $B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. *Let* $S_1, S_2, \ldots S_t \in \{A, B\}$ *with* $S_1 \cdot S_2 \cdots S_t = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. *Then* $a, b, c, d < 2^t$.

Proof. We will induct on t. For the case $t = 1$, note that the values in A and B are at most $1 < 2^1$. Assume that for $l < t$ and $S'_1, S'_2, \ldots, S'_l \in \{A, B\}$ then $S'_1 \cdots S'_l = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$ with $a', b', c', d' < 2^l$.

Next, let $S_1, S_2, \ldots S_t \in \{A, B\}$. Define $S_1 \cdot S_2 \cdots S_{t-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. If $S_t = A$, then

$$MS_t = \begin{bmatrix} a & a+b \\ c & c+d \end{bmatrix} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$$

Otherwise $S_t = B$, and

$$MS_t = \begin{bmatrix} a+b & b \\ c+d & d \end{bmatrix} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$$

In either case, by the inductive hypothesis $a, b, c, d < 2^{t-1}$. Thus, $a + b < 2 \cdot 2^{t-1} = 2^t$, and $c + d < 2 \cdot 2^{t-1} = 2^t$. Therefore, in either case, $a', b', c', d' < 2^t$. □

Lemma 3.2.4. *Let* $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ *and* $B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. *Let*$S_1, S_2, \ldots, S_t \in \{A, B\}$ *with* $t < \log_2(p)$.

Define $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = S_1 \cdot S_2 \cdots S_t$, *with multiplication over* $SL_2(\mathbb{Z}_p)$. *If* $S_t = A$ *then* $a \leq b$, *otherwise if* $S_t = B$ *then* $a > b$.

Proof. Let $S_1, \ldots, S_t \in \{A, B\}$, and $S_1 \cdot S_2 \cdots S_{t-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = M$. By Lemma 3.2.3, no reduction modulo p occurs in the product $S_1 \cdot S_2 \cdots S_t$, as each element of the matrix will be less than $2^t < 2^{\log_2(p)} = p$.

Note if $a = 0$, then $\det(M) \leq 0$. Since $\det(M) = 1$, we must have that $a > 0$.

If $S_t = A$, then

$$M \cdot A = \begin{bmatrix} a & a+b \\ c & c+d \end{bmatrix} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

Thus, since $b \geq 0$, $a' = a \leq a + b = b'$.

If $S_t = B$, then

$$M \cdot A = \begin{bmatrix} a+b & b \\ c+d & d \end{bmatrix} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

Thus, since $a > 0$, $a' = a + b > b = b'$. □

Lemma 3.2.4 allows us to determine a lower bound for the girth of the Cayley graph associated with A_1 and B_1.

Theorem 3.2.5. *The girth of the Cayley graph of* $SL_2(\mathbb{Z}_p)$ *with generators* $\langle A_1, B_1 \rangle$ *is at least* $\log_2(p)$.

Proof. Suppose $S_1 \cdots S_n = T_1 \cdots T_m = M$, with $S_i, T_j \in \{A_1, B_1\}$ and $n, m < \log_2(p)$. Lemma 3.2.4 implies that $S_n = T_m$. By cancelling $S_n = T_m$ from both sides and iterating this argument, we see that $S_{n-i} = T_{m-i}$ and $m = n$, for $0 \leq i < n$.

Therefore, if $S_1 \cdots S_n = T_1 \cdots T_m = M$ are different strings of A_1 and B_1, then n or m must be greater than $\log_2(p)$. □

Using similar arguements, we can conclude that the girths of the Cayleys graphs associated with the generating pairs $\langle A_2, B_2 \rangle$ and $\langle A_3, B_3 \rangle$ are at least $\log_3(p)$.

While not technically a requirement for a secure hash function, a desirable and expected property of hash functions is that the resulting hash values are equally distributed. In [43], Tillich and Zémor claim the following proposition.

Proposition 3.2.6. *If $\mathcal{G}$ is a Cayley graph such that the greatest common divisor of its cycle lengths equals 1, then the distribution of hash values of messages of length n tends to equidistribution when n tends to infinity.*

The proof uses classical graph-theoretic methods by studying the successive powers of the adjacency matrix of the graph. Of particular importance is how quickly equidistribution is achieved. Sufficiently random graphs appear to provide the best results.

3.3 Attacks

The algebraic structure of this hash function can be exploited to find colliding hash messages. Significantly, this hash function is based on matrix multiplication in the special linear group, which is associative. If an attack can find a factorization of the identity matrix of the form $a_1 \cdot a_2 \cdots a_t = I,\ a_i \in \{A, B\}$, then the string of zeros and ones that corresponds to that factorization can be inserted into any message without changing the hash value. Furthermore, the particular choice of generators $\{A, B\}$ has an impact on the structure and can be exploited.

3.3.1 Find Element of Small Order

The group $SL_2(\mathbb{F}_p)$ contains some elements of relatively small order. This attack attempts to find a message which hashes to one of these elements.

The attack begins by computing the hash values of random messages until obtaining a hash value which is a diagonalizable matrix. By [14, Sec. 12.3 Corollary 25], an $n \times n$ matrix M over $\mathbb{F}$ is diagonalizable if its eigenvalues are in $\mathbb{F}$ and its minimal polynomial has no repeated roots.

Suppose m is a bitstring which hashes to a diagonal matrix M. Then M is similar to $D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$, where λ_1, λ_2 are the eigenvalues of M. Note that $D^k = \begin{bmatrix} \lambda_1^k & 0 \\ 0 & \lambda_2^k \end{bmatrix}$, and thus

$\text{ord}(M) = \text{ord}(D) = \text{lcm}(\text{ord}(\lambda_1), \text{ord}(\lambda_2))$.

Since $\lambda_1, \lambda_2 \in \mathbb{F}_p$, their orders will divide $p-1$. Thus, given a factorization of $p-1$ we can easily compute the orders of λ_1 and λ_2. For the values of p which practical for this hash function, factoring $p-1$ is feasible.

Suppose $\text{ord}(M) = d$. Concatenating d copies of the message m will result in a message that hashes to the identity matrix. The concatenation can be inserted into any other message without changing the hash value of the message.

Certainly, however, this attack is only practical if the resulting concatenated message is reasonably short. Therefore, this attack may need to be repeated several times before finding a matrix with a sufficiently small order.

3.3.2 Symmetric Matrices

The particular choice of generators for $SL_2(\mathbb{F}_p)$ can impact the security of the hashing algorithm. Note that in the first two sets of generators of Theorem 3.1.1 A is the transpose of B. This can be exploited in the following manner. Suppose a bitstring $b_1 b_2 \dots b_n$ hashes to a symmetric matrix. Then the message $\overline{b_n} \overline{b_{n-1}} \dots \overline{b_1}$ hashes to the same symmetric matrix, where $\overline{b}$ is the complement of b.

3.3.3 Density Attack

Tillich and Zémor proposed the following attack against the hash function in [42]. Their attack notes the following proposition.

Proposition 3.3.1. *Let $\mathcal{U}$ be the multiplicative monoid generated by $A_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ and $B_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$. Let $M = \begin{bmatrix} a, b \\ c, d \end{bmatrix}$ with $a, b, c, d \in \mathbb{N}$ and $\det(M) = 1$. Then $M \in \mathcal{U}$.*

Before we prove this proposition, let $\mathcal{M}$ denote the set

$$\mathcal{M} = \left\{ M = \begin{bmatrix} a, b \\ c, d \end{bmatrix}, a, b, c, d \in \mathbb{N}, ad - bc = 1 \right\}$$

and note the following lemmas.

Lemma 3.3.2. *Let* $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathcal{M}$. *If* $a+b > c+d$ *and* $a, b > 0$, *then* $a > c$ *and* $b \geq d$. *If* $c+d > a+b$ *and* $c, d > 0$, *then* $c \geq a$ *and* $d > b$.

Proof. Consider the case when $a+b > c+d$ and $a, b > 0$. Suppose $a \leq c$. Then

$$a+b > c+d \Rightarrow b > (c-a)+d \geq d \Rightarrow d-b < 0.$$

But, since $1 = ad-bc \leq cd-bc = c(d-b)$, $d-b = 1 > 0$, which is a contradiction. Therefore, $a > c$.

Suppose $b < d$. Then $a > c+(d-b) \geq c$. Thus, $1 = ad-bc > ad-bc = b(a-c)$. However, $b(a-c) \geq 1$, since b and $a-c$ are positive integers, which is a contradiction. Thus $b \geq d$.

Now consider the case when $c+d > a+b$. Suppose $a > c$. Then $d > (a-c)+b > b$. Also, $1 = ad-bc > cd-bc = c(d-b)$. However, since c and $d-b$ are positive integers, $c(d-b) \geq 1$, which is a contradiction. Thus, $a \leq c$.

Suppose $b \geq d$. Then

$$c+d > a+b \Rightarrow c > a+(b-d) \geq a.$$

Also, $1 = ad-bc \leq ab-bc = b(a-c)$. However, this implies that $a-c > 0 \Leftrightarrow a > c$, which is a contradiction. Thus, $b < d$. □

Lemma 3.3.3. *Let* $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathcal{M}$. *Then* $a > c$ *if and only if* $b \geq d$.

Proof. ($\Rightarrow$) Suppose $a > c$. Then $1 = \det(M) = ad-bc > cd-bc = c(d-b)$. Since $c \geq 0$, we must have that $d-b \leq 0$, and thus $d \leq b$.

($\Leftarrow$) Suppose $b \geq d$. Then $1 = \det(M) = ad-bc \leq ad-bc = b(a-c)$. Since $b \geq 0$, we must have that $a-c > 0$, and thus $a > c$. □

We want to prove that any element of $\mathcal{M}$ can be written as a product of A_1s and B_1s. Choose an arbitrary matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathcal{M}$. We will proceed by cases.

Case 1: $a+b \geq c+d$**.** We will induct on $a+b$. Note that if $a=0$ or $d=0$, then $\det(M) \leq 0$, a contradiction. Thus, when $a+b=1$, it must that $a=d=1$ and $b=c=0$ and therefore $M=I \in \mathcal{U}$.

Now, assume that $M' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix} \in \mathcal{M}$ when $a'+b' \geq c'+d'$ and $a'+b' < a+b$.

If $b=0$, then $1=\det(M)=ad$, which implies that $a=d=1$ and therefore $M=B_1^c \in \mathcal{U}$. Thus, combined with the statement above, we may assume without loss of generality that $a,b,d>0$. Now we must consider whether $a>b$.

Case 1a: $a>b$ Then $a=bq+r$ for $q,r \in \mathbb{N}, 0 \leq r < b$. Define

$$M' = \begin{bmatrix} a-bq & b \\ c-dq & d \end{bmatrix} = MB_1^{-q} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

If $a' = a-bq = 0$, then $a=bq$. Since $\gcd(a,b)=1$, we must have that $b=1$. Since $d>0$, by Lemma 3.3.2, $1=b \geq d > 0$, so $d=1$. Then, since $1=\det(M')=a'd'-b'c'=-c'$, $c'=1$. Thus,

$$M' = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} = A_1B_1^{-1} = MB_1^{-q}$$

and thus $M = A_1B_1^{q-1} \in \mathcal{U}$.

Otherwise $a' = a-bq > 0$. Since $d'=d>0$, we have that $1=\det(M)=a'd'-b'c' \geq -bc'$. Thus $1 \leq bc'$ and since $b \geq 0$, we must have that $c'>0$. Thus $M' \in \mathcal{M}$ and by Lemmas 3.3.2 and 3.3.3, $b'=b \geq d'=d$ and thus $a'>c'$. Thus $a'+b' \geq c'+d'$. Therefore, by the inductive hypothesis, $M' \in \mathcal{U}$ so $M=M'B_1^q \in \mathcal{U}$.

Case 1b: $b>a$ Then $b=aq+r$ for $q,r \in \mathbb{N}, 0 \leq r < a$. Define

$$M' = \begin{bmatrix} a & b-aq \\ c & d-cq \end{bmatrix} = MA_1^{-q} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

If $b' = b-aq = 0$, then since $\gcd(a,b)=1$, we must have that $a=1$. By Lemma 3.3.2, $1=a>c \geq 0$, so $c=0$. Since $1=\det(M')=a'd'-b'c'=d'$, we have that $M'=I=MA_1^{-q}$, so $M=A_1^q \in \mathcal{U}$.

Otherwise $b' = b-ad > 0$. Note that $c'=c \geq 0$. Thus, $1=\det(M')=a'd'-b'c' \leq a'd'$. Since $a'=a>0$, we must have that $d'>0$. Thus, $M' \in \mathcal{M}$, and by Lemma 3.3.3, since

$a' = a > c = c'$, $b' \geq d'$. Thus $a' + b' \geq c' + d'$, and by the inductive hypothesis $M' \in \mathcal{U}$ so $M = M'A_1^q \in \mathcal{U}$.

Case 2: $c + d \geq a + b$**.** We will induct on $c + d$. Again note that if $a = 0$ or $d = 0$, then $\det(M) \leq 0$, a contradiction. Thus, when $c + d = 1$, $a = d = 1$ and $b = c = 0$, which means that $M = I \in \mathcal{U}$.

Now, assume that $M' = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix} \in \mathcal{M}$ when $c' + d' > a' + b'$ and $c' + d' < c + d$.

If $c = 0$, then $1 = \det(M) = ad$, which implies that $a = d = 1$ and therefore $M = A_1^b$. Thus, combined with the statement above, we may assume without loss of generality that $c, d, a > 0$. Now we must consider whether $c > d$.

Case 2a: $c > d$ Then $c = dq + r$ for $q, r \in \mathbb{N}, 0 \leq r < d$.Define

$$M' = \begin{bmatrix} a - bq & b \\ c - dq & d \end{bmatrix} = MB_1^{-1} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

Suppose $c' = c - dq = 0$. Then since $\gcd(c, d) = 1$, it must be that $d = 1$. By Lemma 3.3.2, $1 = d > b \geq 0$, so $b' = b = 0$. Thus, $1 = \det(M') = a'd' - b'c' = a'$, and therefore $M' = I$, so $M = B_1^q \in \mathcal{U}$.

Otherwise $c' > 0$. Since $b' = b \geq 0$, we have that $1 = \det(M') = a'd' - b'c' \leq a'd'$. Since $d' = d > 0$, we must have that $a' > 0$. So, $M' \in \mathcal{M}$ and by Lemmas 3.3.2 and 3.3.3, $d' = d \geq b = b'$ so $c' \geq a'$. Thus $c' + d' \geq a' + b'$ so by the inductive hypothesis $M' \in \mathcal{U}$ and therefore $M = M'B_1^q \in \mathcal{U}$.

Case 2b: $c < d$ Then $d = cq + r$ for $q, r \in \mathbb{N}, 0 \leq r < c$. Define

$$M' = \begin{bmatrix} a & b - aq \\ c & d - cq \end{bmatrix} = MA_1^{-q} = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}.$$

Suppose $d' = d - cq = 0$. Then since $\gcd(c, d) = 1$, we must have that $c = 1$. By Lemma 3.3.2, $1 = c \geq a > 0$, so $a = 1$. Thus, $1 = \det(M') = a'd' - b'c' = -b'$. Thus,

$$M' = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} = B_1A_1^{-1} = MA_1^{-q}$$

and therefore $M = B_1A_1^{q-1} \in \mathcal{U}$.

Otherwise $d' > 0$. Note that $a' = a > 0$. Thus, $1 = \det(M') = a'd' - b'c' \geq -b'c'$, or $1 \leq b'c'$. Since $c' = c \geq 0$, $b' > 0$. Thus, $M' \in \mathcal{M}$, so by Lemmas 3.3.2 and 3.3.3, $a' = a \leq c = c'$ so $b' < d'$. Thus $c' + d' \geq a' + d'$, so by the inductive hypothesis $M' \in \mathcal{U}$ so $M = M'A_1^q \in \mathcal{U}$.

Thus completes the proof of Proposition 3.3.1.

The proof of Proposition 3.3.1 shows that given an element of $\mathcal{M}$, it is easy to find a factorization of that element into a string of A_1s and B_1s by simply applying the recursive algorithm provided in the proof. Essentially this algorithm is applying the Euclidean algorithm on (a, b) if $a + b \geq c + d$, otherwise (c, d), and the quotients from the algorithm are the powers of A_1 and B_1 in the factorization.

Example 3.3.4. Consider $M = \begin{bmatrix} 31 & 9 \\ 24 & 7 \end{bmatrix}$. The following example performs the recursive algorithm given above on M.

$$\begin{bmatrix} 31 & 9 \\ 24 & 7 \end{bmatrix} B_1^{-3} = \begin{bmatrix} 4 & 9 \\ 3 & 7 \end{bmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} 31 & 9 \\ 24 & 7 \end{bmatrix} = \begin{bmatrix} 4 & 9 \\ 3 & 7 \end{bmatrix} B_1^3$$

$$\begin{bmatrix} 4 & 9 \\ 3 & 7 \end{bmatrix} A_1^{-2} = \begin{bmatrix} 4 & 1 \\ 3 & 1 \end{bmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} 4 & 9 \\ 3 & 7 \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ 3 & 1 \end{bmatrix} A_1^2$$

$$\begin{bmatrix} 4 & 1 \\ 3 & 1 \end{bmatrix} B_1^{-4} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} 4 & 1 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} B_1^4$$

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} = A_1 B_1^{-1}$$

Thus, we see that $A_1 B_1^{4-1} A_1^2 B_1^3 = M$.

The purpose of this attack, once again, is to find a factorization of the identity matrix using A_1, B_1, using the above propositions. To do this, the attacker looks for a matrix with non-negative coefficients in $SL_2(\mathbb{Z})$ that is the identity matrix modulo p. By the above propositions, a factorization into A_1, B_1 can be found, which yields a factorization in $SL_2(\mathbb{F}_p)$.

The difficulty is finding a matrix of the form

$$M = \begin{bmatrix} 1 + k_1 p & k_2 p \\ k_3 p & 1 + k_4 p \end{bmatrix}$$

that has determinant 1. This is equivalent to finding a solution to the equation

$$(k_2 k_3 - k_1 k_4)p = k_1 + k_4$$

for (k_1, k_2, k_3, k_4).

To find such a solution, Tillich and Zémor proposed the following algorithm in [42]. Choose a small integer c. We can find a solution to the above equation by searching for a particular solution such that $k_1 + k_4 = cp$. To obtain this particular solution, choose a random prime p' of the same magnitude as p. Take $k_3 = p'$. Next, we must solve the equation

$$k_2 p' = c + cpk_1 - k_1^2.$$

To do this, we must solve the congruence equation $k_1^2 - cpk_1 - c = 0 \pmod{p'}$. Thus, the discriminant $c^2p^2 + 4c$ must be a quadratic residue modulo p'. This occurs with probability $1/2$. If the discriminant is not a quadratic residue, try different values of c and p' until c^2p^2+4c is a quadratic residue. Once this is found it provides the following solution

$$\begin{cases} k_1 = \frac{cp+\sqrt{c^2p^2+4c}}{2} \\ k_2 = \frac{c+cpk_1-k_1^2}{p'} \\ k_3 = p' \\ k_4 = cp - k_1 \end{cases}$$

where $\sqrt{x}$ denotes the positive representation of a square root of x modulo p', if x is a quadratic residue modulo p'.

Example 3.3.5. Let $p = 19$. We will find a matrix which hashes to the identity modulo p. After a small number of trials, we discovered that if $c = 5$ and $p' = 17$, then $c^2p^2+4c = 1 = 16^2 \pmod{p'}$.

Then $k_1 = (5 \cdot 19 + 16)2^{-1} = 13 \pmod{13}$ and using the above formulas, $k_2 = 63, k_3 = 17$ and $k_4 = 82$. The resulting matrix,

$$M = \begin{bmatrix} 1 + k_1 p & k_2 p \\ k_3 p & 1 + k_4 p \end{bmatrix} = \begin{bmatrix} 248 & 1197 \\ 323 & 1559 \end{bmatrix}$$

has determinant 1 and reduces to the identity modulo 19.

To factor M, note that the Euclidean Algorithm on $(1559, 323)$ yields the quotients (in-order):

$$\{4, 1, 4, 1, 3, 3, 4\}$$

Since $c = 323 < 1559$, the first of these factors gives a power of A_1. Using these to implicitly perform the recursive algorithm given in the proof of Proposition 3.3.1, we see that

$$\begin{bmatrix} 248 & 1197 \\ 323 & 1559 \end{bmatrix} A_1^{-4} B_1^{-1} A_1^{-4} B_1^{-1} A_1^{-3} B_1^{-3} A_1^4 = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} = B_1 A_1^{-1}$$

and so

$$M = B_1 A_1^{4-1} B_1^3 A_1^3 B_1 A_1^4 B_1 A_1^4 = I.$$

This attack allows an individual to find a message which hashes to the identity matrix, which in turn can be inserted into any other message to obtain a collision. It could also be used to find primages of matrices whose determinants calculated over $\mathbb{Z}$ are 1. Even if a matrix $M \in SL_2(\mathbb{F}_p)$ does not have determinant 1 when calculated over $\mathbb{Z}$, it should be possible to modify the probabilistic algorithm provided above to find a matrix in $SL_2(\mathbb{Z})$ that reduces to M modulo p. As such, this proposed hash function should not be considered one-way.

This attack was possible because the matrices A_1, B_1 generate the multiplicative monoid of 2×2 matrices over the natural numbers with determinant 1. This is not true for the other generator pairs, $\langle A_2, B_2 \rangle$ and $\langle A_3, B_3 \rangle$. However, note that $A_2 = A_1^2$ and $B_2 = B_1^2$. Thus, if an individual performs the attack above using A_1 and B_1 and obtains a factorization with all even powers, then that provides a factorization using A_2 and B_2. The likelihood that one run of this attack would yield such a factorization is small, but an attacker could try this attack many

times using different random parameter choices until such a factorization is found. Similarly, $A_3 = A_1$ and $B_3 = A_1B_1$, so a factorization involving B_1s to only the power one would yield a factorization using A_3 and B_3.

3.4 Summary

In this chapter we reviewed a hash function proposed and analyzed by $\mathbb{Z}$'emor and Tillich in [48, 42]. The hash function operates in the group $SL_2(\mathbb{F}_p) = \langle A, B \rangle$. It regards a binary message as a string of As and Bs and multiplies the elements of this string to obtain the message digest. The algebraic structure offers little guaranteed protection against attacks.

In Chapter 3.3.3 we showed how one can easily obtain a factorization of the identity matrix, which then allows for the creation of infinitely many hash collisions. It may also be possible to use the attack to find preimages. Increasing the value of p offers little protection against these attacks and thus this hash function cannot be made to offer an adequate level of security for cryptographic applications.

The particular choice of the generating pair $\langle A, B \rangle$ for $SL_2(\mathbb{F}_p)$ can affect the hash function's resistance to attacks which find a factorization of the identity matrix. It appears to be difficult, although not impossible, to use the attack discussed in Chapter 3.3.3 when A and B do not generate all matrices of determinant one over the natural numbers. However, the attack suggests a weakness in the hash function regardless of the specific generators used.

CHAPTER 4. A HASH FUNCTION BASED ON $SL_2(\mathbb{F}_{2^n})$

4.1 Hash Function Construction

At Crypto '94 Jean-Pierre Tillich and Gilles Zémor proposed a modified version of the hash function described in Chapter 3, this time based on on the group $SL_2(\mathbb{F}_{2^n})$. To distinguish between the two hash functions we will refer to the previous hash function as the Characteristic-p hash function and this as the Characteristic-2 hash function. To define this hash function, first let α be the root of an irreducible polynomial $p(x)$ of degree n over $\mathbb{F}_2$.

Theorem 4.1.1. *Let* $A = \begin{bmatrix} \alpha & 1 \\ 1 & 0 \end{bmatrix}$ *and* $B = \begin{bmatrix} \alpha & \alpha+1 \\ 1 & 1 \end{bmatrix}$. *Then* $\langle A, B \rangle = SL_2(\mathbb{F}_{2^n})$.

For proofs of this theorem we refer the reader to [43, 1].

Once again, the hash function is defined by the following. Let $m = b_0 b_1 \dots b_n$ be a binary string. Then the map $H : \{0,1\}^k \to SL_2(\mathbb{F}_{2^n})$ defined by $m \mapsto \pi(b_0) \cdot \pi(b_1) \cdots \pi(b_n)$ where

$$\pi(b) = \begin{cases} A & \text{for } b = 0 \\ B & \text{for } b = 1 \end{cases}$$

is the hash function. In this case, computations are made over the field $\mathbb{F}_{2^n} \cong \mathbb{F}_2[x]/p(x)$.

The hash function could trivially be extended to operate over $SL_2(\mathbb{F}_{p^n})$ for any prime p. The field of order 2^n is the most suitable choice for implementation, as elements of that field are easily represented as bit-strings of length n and computations are the fastest. In [37], Silverman describes a method for computing multiplication of elements in $\mathbb{F}_{2^n}$ with complexity $n+1$. This effectively creates a $3n$-bit hash function, as the fourth element of a matrix can be easily determined given the other three elements.

4.2 Security Properties

The construction of this hash function is identical to that of the Characteristic-p hash function described in Chapter 3. As such, the girth of the Cayley graphs provides a lower bound of security against local modifications of messages yielding collisions. The following theorem establishes the girth for this Cayley graph.

Theorem 4.2.1. *The girth of the Cayley graph associated with the group $SL_2(\mathbb{F}_{2^n})$ and generators $\{A, B\}$ is greater than n.*

To prove this theorem, first consider the following lemmas.

Lemma 4.2.2. *Let $b_1, b_2, \ldots, b_k \in \{A, B\}$ and $k < n$. Then $b_1 \cdots b_k$ has the form M_A when $b_k = A$ or M_B when $b_k = B$, where*

$$M_A = \begin{bmatrix} a_k(\alpha) & b_{k-1}(\alpha) \\ c_{k-1}(\alpha) & d_{k-2}(\alpha) \end{bmatrix}, M_B = \begin{bmatrix} a_k(\alpha) & b_k(\alpha) \\ c_{k-1}(\alpha) & d_{k-1}(\alpha) \end{bmatrix}$$

and a_i, b_i, c_i, d_i are polynomials of degree i over $\mathbb{F}_2[x]$.

Proof. We will induct on k. The base case can easily be verified by considering the elements of A and B. Suppose the lemma holds for strings of length $l < k$. Next, let $b_1, \ldots, b_l \in \{A, B\}$.

Suppose $b_l = A$. By the inductive hypothesis the product $b_1 \cdots b_l \cdot A$ has the form

$$\begin{bmatrix} a_l(\alpha) & b_{l-1}(\alpha) \\ c_{l-1}(\alpha) & d_{l-2}(\alpha) \end{bmatrix} * A = \begin{bmatrix} \alpha a_l(\alpha) + b_{l-1}(\alpha) & a_l(\alpha) \\ \alpha c_{l-1}(\alpha) + d_{l-2}(\alpha) & c_{l-1}(\alpha) \end{bmatrix}.$$

Similarly, the product $b_1 \cdots b_l \cdot B$ has the form

$$\begin{bmatrix} a_l(\alpha) & b_{l-1}(\alpha) \\ c_{l-1}(\alpha) & d_{l-2}(\alpha) \end{bmatrix} * B = \begin{bmatrix} \alpha a_l(\alpha) + b_{l-1}(\alpha) & (\alpha+1) a_l(\alpha) + b_{l-1}(\alpha) \\ \alpha c_{l-1}(\alpha) + d_{l-2}(\alpha) & (\alpha+1) c_{l-1}(\alpha) + d_{l-2}(\alpha) \end{bmatrix}.$$

As $l < k < n$, no reduction occurs modulo the irreducible polynomial of degree n. Thus, in both cases this follows the form stated in the lemma. In the case where $b_l = B$ the same process can be used to show that the product $b_1 \cdots b_l \cdot A$ has the form of M_A and $b_1 \cdots b_l \cdot A$ has the form of M_B. □

Lemma 4.2.3. *Let $S = s_1, s_2, \ldots, s_k$ and $T = t_1, t_2, \ldots, t_l$ be two different strings of As and Bs with $k, l < n$. Then the $s_1 \cdots s_k \neq t_1 \cdots t_k$*

Proof. By the previous lemma, these strings can only have the same form if $k = l$ and $s_k = t_l$. By canceling s_k from both sides and iterating this argument, we see that s_i must equal t_i for all $1 \leq i \leq k$. □

Thus, to see that the girth of the Cayley graph associated with $SL_2(\mathbb{F}_{2^n})$ is at least n, let $s_1, \ldots, s_k$ and $t_1, \ldots, t_l$ be from $\{A, B\}$ and $l, k < n$. By Lemma 4.2.3, the products $s_1 \cdots s_k$ and $t_1 \cdots t_l$ must be different. Therefore the girth of the graph must be at least n.

4.3 Attacks

4.3.1 Discrete Log Attack

In 1996 Willi Geiselmann proposed an algorithm in [17] for finding collisions in this hashing scheme. His attack is based on embedding the rings generated by A and B into finite fields.

Geiselmann's attack provides a form for all powers of A (and B respectively). In this section we will obtain the form for powers of A and the form for the powers of B can be obtained analogously.

4.3.1.1 Mathematical Basis

Recall that $A = \begin{bmatrix} \alpha & 1 \\ 1 & 0 \end{bmatrix}$. The minimal and characteristic polynomial for A over $\mathbb{F}_{2^n}$ is given by $\hat{A}(x) = x^2 + \alpha x + 1$. There exists a ring homomorphism $\phi : \mathbb{F}_{2^n}[x] \to M_2(\mathbb{F}_{2^n})$ such that $\phi(x) = A$, so $\phi(g(x)) = g(A)$.

Note that $\ker \phi = \langle \hat{A}(x) \rangle$. Therefore $\mathbb{F}_{2^n}[x]/\langle \hat{A}(x) \rangle \cong \operatorname{im} \phi$. As ϕ is a ring homomorphism, $\operatorname{im} \phi \leq M_2(\mathbb{F}_{2^n})$. And since $\deg \hat{A}(x) = 2$, $\operatorname{im} \phi = \{\lambda A + \mu I : \lambda, \mu \in \mathbb{F}_{2^n}\} = V$. Thus, all powers of A are in the subring V and have determinant 1. However, V can contain elements other than powers of A. We want to determine when an element of V is a power of A.

To do this, let β be a root of $\hat{A}(x)$ in its splitting field E. The Jordan canonical form of A is given by $\begin{bmatrix} \beta & 0 \\ 0 & \beta^{-1} \end{bmatrix}$ and thus there exists a similarity matrix $P \in GL_2(E)$ such that $A_1 = PAP^{-1} = \begin{bmatrix} \beta & 0 \\ 0 & \beta^{-1} \end{bmatrix}$. Note that the orders of A and A_1 are equal, and say $\text{ord}(A) = d$. Furthermore, note that this implies that $\text{ord}(\beta) = d$ in E^*.

Let $M = \lambda A + \mu I \in V$. Then M has the same order as

$$M_1 = PMP^{-1} = \lambda A_1 + \mu I = \begin{bmatrix} \lambda\beta + \mu & 0 \\ 0 & \lambda\beta^{-1} + \mu \end{bmatrix}.$$

Certainly if M is a power of A then $M^{\text{ord}(A)} = 1$, and thus $M_1^{\text{ord}(A)} = 1$. This implies that $(\lambda\beta + \mu)^{\text{ord}(A)} = 1$. This gives us the forward implication of the following theorem.

Theorem 4.3.1. *Let β be a root of $\hat{A}(x) = x^2 + \alpha x + 1$. Then a matrix $M \in M_2(\mathbb{F}_{2^n})$ is a power of A if and only if the following conditions hold:*

1. *$M = \lambda A + \mu I$ for some $\lambda, \mu \in \mathbb{F}_{2^n}$*
2. *$\det(M) = 1$*
3. *$(\lambda\beta + \mu)^{ord(A)} = 1$*

Proof. All that remains to show is the sufficiency of these three conditions. Suppose the three conditions hold for some $M = \lambda A + \mu I \in V$ where $\lambda, \mu \in \mathbb{F}_{2^n} \leq E$. Then we have that

$$M_1 = PMP^{-1} = \begin{bmatrix} \lambda\beta + \mu & 0 \\ 0 & \lambda\beta^{-1} + \mu \end{bmatrix} = \begin{bmatrix} \gamma & 0 \\ 0 & \delta \end{bmatrix}$$

Because $\det(M) = \det(M_1) = 1$, we have that $\delta = \gamma^{-1}$. By condition 3, $(\lambda\beta + \mu)^d = 1$ and thus $\gamma^d = 1$. As E^* is cyclic, it has a unique subgroup of order d, namely $\langle\beta\rangle$. Since $\gamma^d = 1$, $\gamma \in \langle\beta\rangle$, and therefore $\gamma = \beta^t$ for some $1 \leq t \leq d$.

Hence,

$$M_1 = \begin{bmatrix} \beta^t & 0 \\ 0 & \beta^{-t} \end{bmatrix} = A_1^t$$

and therefore

$$M = P^{-1}M_1P = P^{-1}A_1^tP = (P^{-1}A_1P)^t = A^t.$$

□

The same argument can be applied to obtain the elements of V which are powers of B.

Theorem 4.3.2. *Let γ be a root of the minimal polynomial for B, $\hat{B}(x) = x^2 + (\alpha+1)x + 1$. Then a matrix $M \in M_2(\mathbb{F}_{2^n})$ is a power of B if and only if the following conditions hold:*

1. *$M = \lambda B + \mu I$ for some $\lambda, \mu \in \mathbb{F}_{2^n}$*
2. *$\det(M) = 1$*
3. *$(\lambda\gamma + \mu)^{ord(B)} = 1$*

4.3.1.2 Attack Algorithm

The proposed discrete log attack works as follows. Let β be of root of the minimal polynomial of A and γ a root of the the minimal polynomial of B. Construct and hash a random message to obtain an element $C \in SL_2(\mathbb{F}_{2^n})$. Assume that $C = A^{i_1}B^{i_2}A^{i_3}$ for some powers $i_1, i_2, i_3 \in \mathbb{N}$. By Theorem 4.3.1, a power of A has the form $\lambda_1 A + \mu_1 I$, and by Theorem 4.3.2 powers of B have the form $\lambda_2 B + \mu_2 I$ where $\lambda_i, \mu_i \in \mathbb{F}_{2^n}$.

Therefore, we can construct the equation

$$A^{i_1}B^{i_2}A^{i_3} = (\lambda_1 A + \mu_1)(\lambda_2 B + \mu_2)(\lambda_3 A + \mu_3) = C$$

which gives four non-linear polynomial equations over $\mathbb{F}_{2^n}$, one for each of the four locations in the matrices. We wish to solve this system for values of λ_i, μ_i, $1 \leq i \leq 3$, but these four equations are not sufficient to solve the system. For additional equations, we can add the three equations for the determinants of the powers of A and B. Specifically, the equations $det(\lambda_1 A + \mu_1 I) = 1, det(\lambda_2 B + \mu_2 I) = 1$,and $det(\lambda_3 A + \mu_3 I) = 1$.

Next, attempt to solve this system of equations. If a solution exists, then according to Theorem 4.3.1, then $\lambda_i A + \mu_i I$ is a power of A if and only if $(\lambda_i \beta + \mu_i)^{\mathrm{ord}(A)} = 1$, for $i = \{1, 3\}$. Similarly, by Theorem 4.3.2, $\lambda_2 B + \mu_2 I$ is a power of B if and only if $(\lambda_2\gamma + \mu_2)^{\mathrm{ord}(B)} = 1$.

Assuming this is the case, then these three factors are powers of A or B. Thus we can solve for the values i_1, i_2, i_3 by finding the discrete logarithms of the matrices $\lambda_i A + \mu_i I$, $i = \{1, 3\}$ to the base A, and the discrete logarithm of $\lambda_2 B + \mu_2 I$ to the base B. These can be found by calculating the discrete logarithms of $\lambda_i \beta + \mu_i$, $i = \{1, 3\}$ to the base β and $\lambda_2 \gamma + \mu_2$ to the base γ.

Note that β and γ lie in either $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$, depending on whether $\hat{A}(x)$ and $\hat{B}(x)$ are irreducible over $\mathbb{F}_2(\alpha)$. If one or both of these elements are in $\mathbb{F}_{2^{2n}}$, then the corresponding elements of the solution λ_i, μ_i are regarded as elements of $\mathbb{F}_{2^{2n}}$ and the discrete logarithm is computed in that field.

Note that this attack can be further generalized to find factorizations (and thus collisions) of the form $A^{i_1} B^{i_2} A^{i_3} B^{i_4} \cdots$ of length greater than 3. In the attack description above, note that the system involved 7 equations and 6 unknowns. A fourth unknown matrix $\lambda_4 B + \mu_4 I$ could be added to the system along with the determinant equation. Also, rather than checking the conditions $(\lambda_i \beta + \mu_i)^{\mathrm{ord}(A)} = 1$ and $(\lambda_j \gamma + \mu_j)^{\mathrm{ord}(B)} = 1$ after finding a solution, these conditions could be added to the system of equations. This would potentially allow for factorizations with any length of unknown matrices.

4.3.1.3 Solving Multivariate Polynomial Systems over Finite Fields

The attack described above involves solving a small system of non-linear multivariate polynomials over a finite field. This is a non-trivial task, but there does exist methods for accomplishing this. Typically, Gröbner bases are used to reduce the system into a more manageable form.

Let $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a system of polynomial equations over a field $\mathbb{F}$. The set of solutions to this system is given by

$$V = \{a = (a_1, \ldots a_n) \in \mathbb{F}^n \mid f_1(a) = f_2(a) = \cdots = f_m(a) = 0\}.$$

If I is the ideal generated by $f_1, \ldots, f_m$, then [25, Chapter 5.9] notes that V is also given by

$$V = \{a = (a_1, \ldots a_n) \in \mathbb{F}^n \mid f(a) = 0 \text{ for all } f \in I\}.$$

Therefore, if $g_1, \ldots, g_r \in \mathbb{F}[x_1, \ldots x_n]$ is another generating set for I, then the solutions for $g_1 = \cdots = g_r = 0$ are the same as the solutions for $f_1 = \cdots = f_m = 0$. In this case, we are interested in obtaining a generated set for I of a particular form, specially a Gröbner basis.

Gröbner bases are constructed dependent on an ordering of the monomial terms in polynomials. Before we provide a definition of a Gröbner basis, we will first introduce monomial orderings and leading terms. Dummit and Foote define a monomial ordering as follows [14].

Definition 4.3.3. A monomial ordering is a well ordering $\geq$ on the set of monomials that satisfies $mm_1 \geq mm_2$ whenever $m_1 \geq m_2$ for monomials m, m_1, m_2.

Typically, monomial orderings are constructed lexicographically using an ordering on the variables $x_1, x_2, \ldots, x_n$. If we use the ordering $x_1 > x_2 > \cdots > x_n$, a monomial $Ax_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$ has a higher order than $Bx_1^{b_1}x_2^{b_2}\cdots x_n^{b_n}$ if $(a_1, a_2, \ldots, a_n)$ is lexicographically greater than $b_1, b_2, \cdots, b_n$. The *leading term* of a polynomial f is the monomial term in f of maximal order. With this, we can provide the definition of a Gröbner basis.

Definition 4.3.4. [14, Section 9.6] A Gröbner basis for an ideal I in $\mathbb{F}[x_1, \cdots, x_n]$ is a finite set of generators $\{g_1, \ldots, g_m\}$ for I whose leading terms generate the ideal of all leading terms of I.

Gröbner bases can be computed using Buchberger's Algorithm [5, 14, 25]. If the ideal I is zero-dimensional, then the more efficient FGLM algorithm can be used [15].

To solve a system of polynomial equations $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ we wish to find elements in the ideal $I = \langle f_1, \ldots f_m \rangle$ which contain only some of the variables $x_1, \ldots, x_n$. The following theorem helps us find such elements when they exist.

Theorem 4.3.5. *[14, Section 9.6] Let $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and $I = \langle f_1, \ldots, f_m \rangle$. Suppose $G = \{g_1, \ldots g_r\}$ is a Gröbner basis for I using the ordering $x_1 > x_2 > \cdots > x_n$. Then $G \cap \mathbb{F}[x_{i+1}, \ldots, x_n]$ is a Gröbner basis of the elimination ideal $I_i = I \cap \mathbb{F}[x_{i+1}, \ldots, x_n]$.*

In an ideal case, each elimination ideal I_i is non-zero, which implies that there exists a polynomial $g \in G$ that involves only the polynomials $x_{i+1}, \ldots, x_n$. Thus we could use $g \in \mathbb{F}[x_n]$

to solve for x_n, and back-substitute to solve for x_{n-1} using $g' \in \mathbb{F}[x_{n-1}, x_n]$, repeating this process of back-substitution until a complete solution is found.

Gröbner bases are not always successful at solving systems of multivariate polynomial equations. Recently, Ding, Gower and Schmidt developed a new way to solve systems of polynomial equations over finite fields [13]. Their method involves lifting the problem to an extension field where they system becomes a single-variable equation. Then it is possible to apply the known methods for solving univariate equations over finite fields. For instance, the Berlekamp algorithm [4] can factor polynomials if the degree is relatively small.

4.3.1.4 Example

We implemented the attack using the Axiom computer algebra system [33]. The code for this attack can be found in Appendix A. Based on this implementation, we have found that the collisions discovered by this attack are highly impractical, which the author freely admits in [17].

For example, using the Axiom implementation found in Appendix A, we discovered the following collision when this hash calculated over $\mathbb{F}_{2^{60}}$ represented as $\mathbb{F}_2[x]/\langle x^{60} + x + 1\rangle$.

$$A^{160} * B^{258} = A^{22367311414644889} B^{1152921504606846719} A^{22367311414644729}$$

In particular, note that in this instance

$$\text{ord}(A) = \text{ord}(B) = 1152921504606846977 = 2^{60} + 1$$

4.3.1.5 Practicality of Attack

In practice, the powers on A and B discovered in the attack are usually roughly equal to the orders of A and B in magnitude. The example above used the field $\mathbb{F}_{2^{60}}$ whereas a more realistic implementations of this hash function would use $\mathbb{F}_{2^n}$ where n is at least 120. In this case, the orders of A and B will divide either $2^{120}+1$ or $2^{120}-1$. As the orders of A and B are dependent on not only the degree but also the choice of the irreducible polynomial, it would

be possible calculate the orders of A and B using different irreducible polynomials of degree n until their orders are $2^n + 1$ or $2^n - 1$.

Even with $n = 60$, the example above found a collision involving approximately 1.2×10^{18} bits. To put this number into perspective, this is roughly a message of size $132,000$ terabytes, hardly a reasonable message size. This attack algorithm, in general, would find an even longer message in order to obtain collisions using a field with a more realistic value of n.

However, it should be noted that this attack is surprisingly efficient, relative to most attacks against cryptographic systems. There are two computationally expensive operations in this attack. The first is to solve the system of non-linear polynomial equations over the field. However, since the system is small and the polynomials are of small degree it is relatively easy to compute the Gröbner basis and find a solution if one exists.

The complexity of the attack is instead dominated by the calculations of the discrete logarithms in either $\mathbb{F}_{2^n}$ or $\mathbb{F}_{2^{2n}}$. Calculating discrete logarithms is generally considered a hard problem. In fact, many cryptosystems are based on the difficulty of calculating discrete logarithms. However, computing discrete logarithms in fields of characteristic p is much easier when p is small. Coppersmith developed efficient methods for calculating discrete logarithms such fields, creating a variant of the index-calculus method that runs in time $\exp(n^{1/3})$ in fields $\mathbb{F}_{2^n}$ [8, 9]. Other methods using the function field sieve that claim to be even faster [22]. Thomé was able to compute discrete logarithms in the field $\mathbb{F}_{2^{607}}$ using a variation of Coppersmith's algorithm and believe that computations in fields as large as $\mathbb{F}_{2^{9}97}$ are within reach of well-funded institutions [41].

Based on this, it is not possible to choose a large enough value for n such that computing the discrete logarithms is prohibitively difficult without making the hash function impractical. Nonetheless, it is a simple matter to make this step as difficult as possible. Recall that if the minimal polynomials for A and B are irreducible over $\mathbb{F}_2(\alpha)$, then the discrete logarithms are computed in the field $\mathbb{F}_{2^{2n}}$. Thus, by choosing the irreducible polynomial carefully we can make this attack harder without much difficulty or reduction in efficiency.

Because the message lengths of colliding messages are extremely long it is unlikely that this

attack could be used in practice to compromise the this hash function. However, the existence and efficiency of this attack does seriously question the security of the Characteristic-2 hash function.

4.3.2 Small Order Attack

At the Crypto 2000 conference, Steinwandt, Grassl, Geiselmann and Beth described an attack against the Tillich-Zémor scheme that obtains a factorization of the identity matrix by finding an element of $SL_2(\mathbb{F}_{2^n})$ of small order [38]. Once a factorization of the identity matrix is found as a product of A's and B's an unlimited number of messages with colliding hash values can be found.

4.3.2.1 Mathematical Basis

The attack is based on the fact that there are elements of $SL_2(\mathbb{F}_{2^n})$ that have small orders. First consider the following property about the order of elements from this group.

Theorem 4.3.6. *Let p be a prime and m be a positive integer. Suppose $M \in SL_2(\mathbb{F}_{p^m})$. Then $ord(M)|p^m+1$ or $ord(M)|p^m-1$.*

Proof. Let $g(x) = x^2 + ax + b$ be the characteristic polynomial for M. Proceed by cases.

Case 1: Suppose $g(x)$ is irreducible over $\mathbb{F}_{p^m}$. Then $g(x) = (x-\alpha)(x-\beta) \in E[x]$, where E is the splitting field $\mathbb{F}_{p^m}(\alpha) = \mathbb{F}_{p^{2m}}$. By considering the Frobenius automorphism, note that $\beta = \alpha^{p^m}$, since the only non-trivial member of $Gal(E/\mathbb{F}_{p^m})$ maps $x \mapsto x^{p^m}$. Also, the determinant of M is equal to the product of the eigenvalues of M. Thus $\alpha\beta = \alpha^{p^m+1} = 1$.

Since M is similar to $\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$ in E, we have that $\text{ord}(M) = \text{ord}(\alpha)|p^m+1$.

Case 2: Suppose $g(x)$ is reducible over $\mathbb{F}_{p^m}$. Then $g(x) = (x-\alpha)(x-\beta)$ where $\alpha, \beta \in \mathbb{F}_{p^m}$. Again, the determinant of M is the product of the eigenvalues, so $\alpha\beta = 1$ and thus $\beta = \alpha^{-1}$. Since $\alpha \in \mathbb{F}^*_{p^m}$, $\text{ord}(\alpha) = \text{ord}(\beta)|p^m-1$.

Note that M is similar to either $\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$ or $\begin{bmatrix} \alpha & 0 \\ 1 & \alpha \end{bmatrix}$ depending on whether the eigenvalues α, β are distinct.

If $M \sim \begin{bmatrix} \alpha & 0 \\ 1 & \alpha \end{bmatrix}$, then $\alpha = \pm 1$. In this case, $\text{ord}(M) = p$. Otherwise, $\alpha = -1$, in which case $\text{ord}(M) = \text{lcm}(2, p)$.

If $M \sim \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha^{-1} \end{bmatrix}$, then $\text{ord}(M) = \text{ord}(\alpha) | p^m - 1$. □

By the above theorem, if a matrix $M \in SL_2(\mathbb{F}_{2^n})$ is found whose entries lie in $\mathbb{F}_{2^m}) \leq \mathbb{F}_{2^n}$, the order of M will be bounded above by $2^m + 1$. More generally, if a matrix $M \in SL_2(\mathbb{F}_{2^n}$ is similar to a matrix in $SL_2(\mathbb{F}_{2^m})$, the order of M will be bounded by $2^m + 1$. The trace operation can be used to test whether a given matrix satisfies this property.

Theorem 4.3.7. *Let $M \in SL_2(\mathbb{F}_{p^n})$ and $\mathbb{F}_{p^m} \leq \mathbb{F}_{p^n}$. Then M is similar to a matrix in $SL_2(\mathbb{F}_{p^m})$ if and only if the trace of M is an element of $\mathbb{F}_{p^m}$.*

Proof. The forward implication is obvious from the closure of $\mathbb{F}_{p^m}$ under addition.

Suppose the trace of M is in $\mathbb{F}_{p^m}$. Let $g(x)$ be the minimal polynomial for M. If $g(x)$ is linear, then $M = \pm I_2 \in SL_2(\mathbb{F}_p)$.

If $g(x)$ is not linear, then $g(x) = x^2 - \text{trace}(M)x + 1$. Suppose that $g(x) = (x-\alpha)(x-\beta) \in \mathbb{F}_{p^n}(\alpha)[x]$. Then M is similar to its Jordan canonical form in $GL_2(\mathbb{F}_{p^n}(\alpha))$, which is $\begin{bmatrix} \alpha & 1 \\ 0 & \alpha \end{bmatrix}$ if $\alpha = \beta$ or $\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$ otherwise.

In the first case, we have that $\alpha^2 = 1$, so $\alpha = \pm 1$. Thus, the Jordan canonical form of M is in $SL_2(\mathbb{F}_p)$.

In the second case, consider the matrix $M' = \begin{bmatrix} \text{trace}(M) & -1 \\ 1 & 0 \end{bmatrix} \in SL_2(\mathbb{F}_{p^m})$ whose characteristic polynomial is $g(x) = (x-\alpha)(x-\beta)$. Since α, β are distinct, M' is similar to $\begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix}$. Thus, M is similar to $M' \in SL_2(\mathbb{F}_{p^m})$. □

As a direct consequence of Theorems 4.3.6 and 4.3.7, if the trace of a matrix $M \in SL_2(\mathbb{F}_{2^n})$ lies in some subfield $\mathbb{F}_{2^m} \leq \mathbb{F}_{2^n}$, then the order of M is bounded above by $2^m + 1$. Furthermore,

if trace$(M) = \alpha$ and $\mathbb{F}_2(\alpha)$ is a small subfield of F_{2^n}, then the order of M will be small.

4.3.2.2 Attack Algorithm

Let $\mathbb{F}_{2^n}$ be represented as $\mathbb{F}_2(\alpha)$ where α is a root of an irreducible polynomial $p(x)$ of degree n. Furthermore, suppose that $p(x)$ is decomposable. That is $p(x) = (f \circ g)(x)$ where $f, g \in \mathbb{F}_2[x]$ and $\deg(f), \deg(g) > 1$. Polynomial decompositions can be found efficiently. For instance, [24] provides a polynomial decomposition algorithm with complexity $O(n^{\log n})$ for polynomials of degree n over fields where a polynomial-time polynomial factorization algorithm exists. For other algorithms see [2, 44, 45, 19].

The attack algorithm calls for finding a matrix $M \in SL_2(\mathbb{F}_{2^n})$ whose trace equals $g(\alpha)$. In this case, trace(M) is a root of $f(x)$, which must be irreducible since $p(x)$ is irreducible. Therefore $\mathbb{F}_2(\text{trace}(M))/\mathbb{F}_2$ is a degree $\deg(f(x))$ extension field.

By Theorems 4.3.6 and 4.3.7, the order of M is no greater than $2^{\deg(f(x))} + 1$. Therefore, if the the matrix M has order d and M is the hash value of a bitstring of length l, then a message of length dl hashes to the identity by repeating the bitstring d times.

The major source of complexity in this attack is finding a message which hashes to such a suitable matrix M. Lemma 4.2.2 states that the trace of the hash of a message of length $k < n$ is a degree k polynomial. This suggests that a search over bitstrings of length $\deg(g(x))$ should yield a message which hashes to a matrix with trace $g(\alpha)$. Thus, to limit the search space it is desirable to construct a decomposition of $p(x)$ where $g(x)$ has a small degree. However, $f(x)$ must also have a small degree, to limit the order of the resulting matrix.

4.3.2.3 Example

The following example was constructed using the Axiom code provided in Appendix 2.

Consider the field $\mathbb{F}_{2^{100}}$ represented as $\mathbb{F}_2(\alpha)$ where α is a root of $p(x) = x^{100} + x^{75} + 1$. Note that $p(x) = (x^4 + x^3 + 1) \circ (x^{25})$.

A search of bitstrings of length 25 discovers that the message

$$1000000000100000010100110$$

hashes to a matrix with trace α^{25}.

Theorem 4.3.6 states that the order of this matrix will less less than $2^4 + 1 = 17$. In this case, the matrix has order 15.

Therefore, if the bitstring 100000000010000001010011 0 is repeated 15 times, the resulting message of $25 \cdot 15$=375 bits will hash to the identity matrix.

4.3.2.4 Practicality of Attack

Unlike the previous attack against the $SL_2(\mathbb{F}_{2^n})$ hashing scheme, this attack is capable of finding relatively short messages with colliding hash values. Furthermore, depending on the decomposition of $p(x)$, the attack can be performed very quickly, as it does not rely on computing discrete logarithms in a large field.

However, this attack requires that the irreducible polynomial which defines the field representation be decomposable. This is extremely rare for practical values of n.

Theorem 4.3.8. *As n increases, the percentage of decomposable irreducible polynomials over $\mathbb{F}_2$ approaches 0.*

To prove this result, first note the following lemma from [26, pg. 142, Ex. 3.26 & 3.27].

Lemma 4.3.9. *Let I_n be the number of irreducible polynomials of degree n over $\mathbb{F}_2$. Then*

$$\frac{2^n - 2^{n/2+1} + 2}{n} \leq I_n \leq \frac{2^n - 2}{n}.$$

First we will find a upper bound on the number of decomposable irreducible polynomials. Let $f(x)$ be an irreducible polynomial of degree n. Suppose $f(x)$ can be decomposed into $(g \circ h)(x)$. Then, $\deg g \cdot \deg h = n$ and $g(x)$ must be irreducible over $\mathbb{F}_2$. Thus, an upper bound on the number of irreducible decomposable polynomials of degree n is the number of pairs $\{g(x), h(x)\}$ such that $g(x)$ is irreducible over $\mathbb{F}_2$ and $\deg g \cdot \deg h = n$.

For each $g(x) \in \mathbb{F}_2[x]$ of degree d where $d|n$, $h(x)$ can be any polynomial of degree n/d, of which there are $2^{n/d}$. Also, by Lemma 4.3.9, the number of irreducible polynomials of degree d is less than $\frac{2^d-2}{d}$. Thus, the number of pairs $\{g(x), h(x)\}$ is less than

$$\sum_{d|n} \frac{2^d - 2}{d} 2^{n/d} = \sum_{d|n} \frac{2^{n/d+d} - 2}{d}.$$

Each term of this summation is less than or equal to $\frac{2^{n/2+2}-2}{2}$. Furthermore, the number of non-trivial divisors of n is less than $\log_2(n)$. Thus

$$\sum_{d|n} \frac{2^{n/d+d}-2}{d} \leq \frac{\log_2(n)}{2}\left(2^{n/2+2}-2\right).$$

By Lemma 4.3.9, a lower bound on the number of irreducible polynomials of degree n is $\frac{2^n-2^{n/2}+2}{n}$. Thus the ratio of irreducible decomposable polynomials of degree n to all irreducible polynomials of degree n over $\mathbb{F}_2$ is less than

$$\begin{aligned}
\frac{\frac{\log_2(n)}{2}\left(2^{n/2+2}-2\right)}{\frac{2^n-2^{n/2}+2}{n}} &= \frac{n\log_2(n)\left(2^{n/2+2}-2\right)}{2\left(2^n-2^{n/2}+2\right)} \\
&\leq \frac{n\log_2(n)\left(2^{n/2+2}-2\right)}{2^{n/4}\left(2^{3n/4}-2^{n/4}+\frac{2}{2^{n/4}}\right)} \\
&\leq \left(\frac{n\log_2(n)}{2^{n/4}}\right)\left(\frac{2^{n/2+2}}{2^{3n/4}-2^{n/4}}\right)
\end{aligned}$$

Using l'Hôpital's rule we can take the limit of $\frac{n\log_2(n)}{2^{n/4}}$ as n approaches infinity.

$$\begin{aligned}
\lim_{n\to\infty} \frac{n\log_2(n)}{2^{n/4}} &= \lim_{n\to\infty} \frac{\ln(n)/\ln(2)}{\frac{1}{4}2^{n/4}\ln(2)} \\
&= \lim_{n\to\infty} \frac{4\ln(n)}{2^{n/4}\ln(2)^2} \\
&= \lim_{n\to\infty} \frac{4/n}{\frac{1}{4}2^{4/n}\ln(2)^3} \\
&= \lim_{n\to\infty} \frac{16}{n2^{n/4}\ln(2)^3} \\
&= 0
\end{aligned}$$

The limit of the other factor, $\frac{2^{n/2+2}}{2^{3n/4}-2^{n/4}}$, can be taken as follows.

$$\lim_{n\to\infty} \frac{2^{n/2+2}}{2^{3n/4}-2^{n/4}} = \lim_{n\to\infty}\left(\frac{2^2}{2^{n/4}}\right)\left(\frac{2^{n/2}}{2^{n/2}-1}\right) = 0\cdot 1 = 0$$

Therefore, the limit of the ratio of decomposable irreducible polynomials of degree n to all irreducible polynomials of degree n approaches 0 as n goes to infinity.

Table 4.1 Decomposable Irreducible Polynomials

Deg.	# Irr.	# Irr. Decomp.	Ratio	Upper Bound	Limit Estimate
9	56	9	3.57×10^{-2}	2.86×10^{-1}	2.432
15	2182	8	3.67×10^{-3}	5.13×10^{-2}	0.499
21	99857	34	3.40×10^{-4}	4.01×10^{-3}	0.116
25	1342176	14	1.04×10^{-5}	1.43×10^{-4}	0.0345

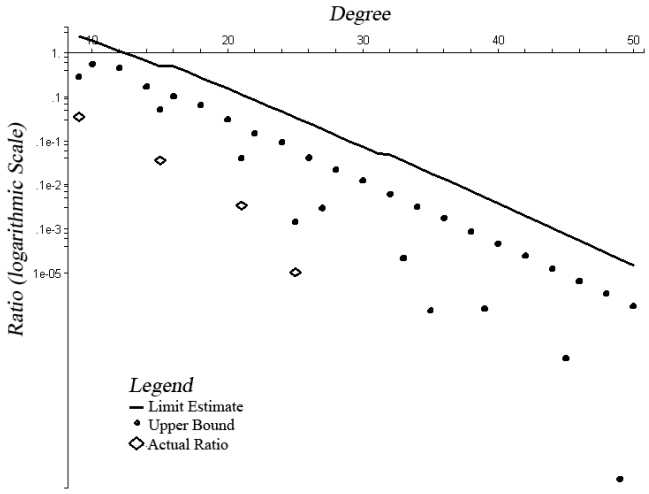

Figure 4.1 Ratio of Decomposable Irreducible Polynomials (Log Scale)

In the case of this attack, the speed at which this ratio goes to 0 is of importance. The ratio used to construct the limit, $\frac{n \log_2(n)\left(2^{n/2+2}-2\right)}{2\left(2^n-2^{n/2+2}\right)}$ is extremely conservative. But at $n = 40$, even this ratio is approximately 0.00038. At $n = 128$ this ratio is 9.7×10^{-17}.

Using the computer algebra system Axiom [33], we can find the exact number of decomposable irreducible polynomials for some small values of n. The polynomial decomposition algorithm used in Axiom is an implementation of Algorithm 4 in [24]. This algorithm cannot decompose a polynomial of degree n if the characteristic of the field divides n.

The results of this program are summarized in Table 4.1. For some small values of n, we determined the ratio of decomposable irreducible polynomials to the total number of irreducible polynomials. We also give a ratio based on the upper bound for decomposable irreducible polynomials, which is the number of pairs $\{g(x), h(x)\}$ such that $g(x)$ is irreducible over $\mathbb{F}_2$ and

$\deg g \cdot \deg h = n$. Finally, we provide a ratio based on the limit calculation, $\frac{n \log_2(n)\left(2^{n/2+2}-2\right)}{2\left(2^n-2^{n/2+2}\right)}$. These values are plotted in Figure 4.1 using a logarithmic scale.

Note that we can use polynomial decomposition algorithms to determine whether a given irreducible polynomial is decomposable. Such polynomials should be avoided when choosing a representation for the field. As the vast majority of irreducible polynomials are indecomposable it is not difficult to find an appropriate polynomial. Another precaution against this attack is to choose a irreducible polynomial with prime degree. In this case there can be no non-trivial polynomial decomposition for $p(x)$.

4.4 Summary

In this chapter we reviewed the Characteristic-2 hash function proposed by Zémor and Tillich in [43]. This hash function is a variation of that given in Chapter 3, operating in $SL_2(\mathbb{F}_{2^n})$ as opposed to $SL_2(\mathbb{F}_p)$. The Characteristic-2 hash function has the property that in order to obtain a collision by modifying only a small section of bits in a message, that section must be at least n-bits long.

However, as discussed in Chapter 4.3, two major attacks have been proposed against this hash function. The first of these attacks creates a system of non-linear multivariate polynomials over $\mathbb{F}_2$ and finds collisions after calculating a small number of discrete logarithms. The second attack requires decomposing the irreducible polynomial used to represent the field $\mathbb{F}_{2^n}$.

Neither of these attacks are capable of mounting a practical attack against the Characteristic-2 hash function. By carefully choosing the irreducible polynomial $p(x)$ representing the field we can make it highly unlikely that these attacks would yield a practical collision. First, $n = \deg(p(x))$ should be prime, making it impossible to find a non-trivial polynomial decomposition. Furthermore, Theorem 4.3.6 implies that if we choose n such that $(2^n+1)(2^n-1) = 3p_1p_2$ for some primes p_1, p_2, then the group $SL_2(\mathbb{F}_{2^n})$ will have a small number of elements of small order. This attempts to prevent attacks which aim to find elements of small order.

Furthermore, when $(2^n + 1)(2^n - 1) = 3p_1p_2$ the orders of A and B will be $2^n + 1$ or $2^n - 1$. Additionally, it is desirable for the minimal polynomials of A and B to be irreducible

over $\mathbb{F}_{2^n} = \mathbb{F}_2(\alpha)$. If this condition is not met for a given irreducible polynomial it should be possible to find a different polynomial where it is met.

While the two attacks presented do not pose an immediate threat to the security of the Characteristic-2 hash function, they do question it. The algebraic structure of the hash function can be exploited to create attacks, but it is not clear if a practical attack can be developed.

CHAPTER 5. VARIATIONS

The hash functions of Chapters 3 and 4 have a common property which is highly useful in attacks. Recall that because matrix multiplication is associative, one way to find collisions in these hash functions is to find a message which hashes to the identity matrix. This chapter discusses two variations of the hash function design which attempt to limit or remove that vulnerability.

5.1 Previous Work

In [36], Shpilrain identifies two weaknesses of the Tillich-Zémor hashing scheme discussed in Chapter 4, which also apply to the scheme discussed in Chapter 3. First, the matrices A and B are invertible. Second, the hash function is based on matrix multiplication which is associative.

Shpilrain proposes a variation of the Tillich-Zémor hash function in [36] that avoids these properties. He does this by simply using elements of the field $\mathbb{F}_{2^n}$ represented as polynomials and defining a non-associative binary operator.

5.1.1 Shpilrain's Variation

Operations in the hash function are performed in the field $\mathbb{F}_{2^n} = \mathbb{F}_2(\alpha)$, where α is a root of an irreducible polynomial $p(x)$ of degree n. Rather than defining two matrices that are the hashes of 0 and 1 bits, Shpilrain's variation defines two field elements $P(\alpha)$, $Q(\alpha) \in \mathbb{F}_2(\alpha)$. Also, fix an additional four elements $u_1(\alpha), u_2(\alpha), v_1(\alpha), v_2(\alpha) \in \mathbb{F}_2(\alpha)$ such that $u_1(\alpha) + v_2(\alpha) \neq u_2(\alpha) + v_1(\alpha)$.

The hash function is built from a binary operation on $\mathbb{F}_{2^n}$, which is defined as follows. Let $f(\alpha), g(\alpha) \in \mathbb{F}_{2^n}$.

$$f(\alpha) \circ g(\alpha) = (f(\alpha) + u_1(\alpha)) \cdot (g(\alpha) + u_2(\alpha)) \cdot f(\alpha) \cdot v_1(\alpha) + g(\alpha) \cdot v_2(\alpha)$$

Shpilrain notes in [36] that this operation is non-associative and non-commutative when $u_1(\alpha) + v_2(\alpha) \neq u_2(\alpha) + v_1(\alpha)$.

A hash function is constructed from this binary operation by splitting a message into fixed length blocks, computing the hash on each individual block, and combining the hash values of the blocks. The binary operation is used in both computing the hashes of each block and combining the block, but in different ways.

A message m is split into blocks $B_1, B_2, \ldots, B_l$ such that blocks $B_1, \ldots B_{l-1}$ each have length k and B_l has length less than or equal to k.

The hash value of an individual block is computing using the following recursive formula $F : \{0,1\}^* \to \mathbb{F}_{2^n}$. Let $B_i = b_1 \cdots b_t$, with $b_i \in \{0,1\}$ and defined F as

$$F(b_1 \ldots b_s) = \begin{cases} \pi(b_1) & \text{if } s = 1 \\ F(b_1 \ldots b_{s-1}) \circ \pi(b_s) & \text{otherwise} \end{cases}$$

where

$$\pi(b) = \begin{cases} P(\alpha) & \text{for } b = 0 \\ Q(\alpha) & \text{for } b = 1 \end{cases}.$$

Next the hash values from each block are combined using the recursive formula $H_1 : \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ defined by

$$H_1(B_1 \ldots B_s) = \begin{cases} F(B_1) & \text{if } s = 1 \\ H_1(F(B_1) \ldots F(B_{s-1})) \circ F(B_s)) & \text{otherwise} \end{cases}.$$

In particular, Shpilrain suggests the following parameters. To create a 163-bit hash function, let

$$p(x) = x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1$$

and $P(\alpha) = \alpha^7 + 1$ and $Q(\alpha) = \alpha^8 + 1$. Furthermore, appropriate choices for the fixed field elements for $\circ$ are

$$u_1(\alpha) = \alpha^2,\ u_2(\alpha) = \alpha, \quad v_1(\alpha) = 1,\ v_2(\alpha) = \alpha + 1.$$

For a short list of examples see [36].

5.1.2 Analysis of Shpilrain's Variation

Certainly the associativity of matrix multiplication creates a vulnerability in Tillich and Zémor's hash functions. Shpilrain's proposal indeed creates a non-associative function, but many details of his design are not clear. In [36] Shpilrain provides no explanation for the construction of the binary operator $\circ$, nor any discussion of the security implications of choices of $P(\alpha)$ and $Q(\alpha)$.

Furthermore, Shpilrain's claim that these scheme holds the same advantages of the Tillich and Zémor scheme is unfounded. The hash functions of Tillich and Zémor relate their security to the Cayley graph of the groups $SL_2(\mathbb{F}_p)$ or $SL_2(\mathbb{F}_{2^n})$. Specifically, they have the property that collisions involving local modifications of text must involve at least as many bits as the girth of the graph. Furthermore, finding collisions is equivalent to finding factorizations of the identity, which they suggest is often hard.

Shpilrain offers no proof of security in this scheme. Collisions in the the hash values of individual message blocks would be related to the Cayley graph of the quasigroup associated with the field and the operation, but it is unclear whether one could prove any properties about the girth of this graph.

On the surface, one would probably expect that Shpilrain's variation would be more difficult to attack, but there is no solid evidence of this. In fact, Donghoon Chang has already developed a preimage attack against this scheme in [6]. Shpilrain's variation thus is not entirely successful at solving the security problems with the basic design of Tillich and Zémor's hash functions.

5.2 Using Left Division

There is an uneasy relationship between algebraic structure and security. If an algorithm has little structure, such as Shpilrain's scheme in 5.1, then it can be difficult to prove security results. However, algorithms with too much structure can often have that structure exploited in attacks, as was in the case with Tillich and Zémor's schemes. Matrix left division is a non-associative binary operation that provides a similar level of structure to that of matrix multiplication.

5.2.1 Hash Function Construction

A naive extension of the Tillich and Zémor hash function construction to left division simply performs matrix left division in place of matrix multiplication. Left division is defined by $x \backslash y = x^{-1} \cdot y$. Note that for a matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in SL_2(\mathbb{F}_{2^n})$, $M^{-1} = \begin{bmatrix} d & b \\ c & a \end{bmatrix}$. Therefore, matrix left division is no more computationally expensive than matrix multiplication.

Define $SL_2(F_{2^n}) = \langle A, B \rangle$ as in Chapter 4, π be the same map from $\{0,1\}$ to $\{A, B\}$ and $m = b_0 b_1 \dots b_k$ be a binary string. Define by hash function $H_1 : \{0,1\}^* \to SL_2(\mathbb{F}_{2^n})$ by

$$H_1(b_0 b_1 \dots b_k) = \pi(b_0) \backslash \pi(b_1) \backslash \cdots \backslash \pi(b_k)$$

However, one quickly is able to see a flaw in this design. The messages 00 and 11 hash to the identity matrix. In particular, concatenating 00 or 11 to the beginning of a message does not change the hash value. To solve this problem, instead of starting the hash value computation with the identity matrix, we start with a matrix C that is not a power of A or B and begin the set of left divisions. An obvious choice for C is $C = AB$. From this we can define our hash function as

$$H_2(b_0 b_1 \dots b_k) = C \backslash \pi(b_0) \backslash \pi(b_1) \backslash \cdots \backslash \pi(b_k).$$

This construction still shares the security result between Cayley graphs and collisions. In Tillich and Zémor's schemes, a collision of two messages could be thought of as two distinct paths on the Cayley graph associated with A and B using matrix multiplication that start at

the identity matrix and end at the same group element. In this case, the graph is now obtained using matrix left division, and the paths start at the matrix C.

5.2.2 Properties

Matrix left division still provides a great deal of algebraic structure. One can quickly see that hash values of message have the following form.

Proposition 5.2.1. *Let $m = b_0 b_1 \dots b_k$ be a binary string and $P = \pi(b_0)\pi(b_2)\cdots$ and $Q = \pi(b_1)\pi(b_3)\cdots$. If k is odd, then*

$$H_2(b_0 b_1 \dots b_k) = Q^{-1}CP$$

else when k is even

$$H_2(b_0 b_1 \dots b_k) = P^{-1}CQ.$$

Note that the Characteristic-p and Characteristic-2 hash functions are parallelizable because matrix multiplication is associative. That is, the computation of the hash function can be split by splitting the message into smaller pieces, calculating the hash value of those pieces, and multiplying the results. Furthermore, the hash values can start being computed before the full message is available.

Because of Proposition 5.2.1, the hash function using left division still has these properties. If we consider a message as a string of As and Bs, the computation of the hash function is equivalent to multiplying the values in even-numbered bits and the values in-odd numbered bits separately. These computations can be performed in parallel before the complete message is given. Once the complete message is processed this way, one of the two products is inverted, and multiplied by C and the other product.

One problem with the Tillich and Zémor hash functions was that once a factorization of the identity matrix is found as a product of A's and B's, an unlimited number of collisions can be found, due to the associativity of matrix multiplication. Unfortunately, Proposition 5.2.1 shows that that also applies to the hash function H_2.

Obtaining a factorization of the identity into A's and B's is equivalent to finding a message which hashes to the identity. Suppose $s_0 \dots s_l$ is a binary message hashing to the identity. We can insert the string $s_0 s_0 s_1 s_1 \cdots s_l s_l$ anywhere in a message m to obtain collision in the hash function. Inserting this string into the message has the effect of inserting $\pi(s_0) \cdots \pi(s_l)$ into P and Q. Since $\pi(s_0) \cdots \pi(s_l)$ equal the identity matrix, it does not change the values of P and Q.

5.2.3 Embedding Message Length

The problem of being able to insert a message which hashes to the identity matrix in order to obtain a collision is analogous to a problem with iterated hash functions. Recall that hash functions using MD-construction encode the the length of the message as the last block of the message, a process known as MD-strengthening.

Iterated hash functions split a message into blocks and iterate over the blocks using a compression function. A general attack against an iterated hash function that does not use MD-strengthening works as follows. Let H' be the iterated hash function and m be some message that is a multiple of the block size of H'. Note that $H'(m) = h'$ is a binary message with a length equal to the block size of H'. Because of the construction of iterated hash functions, $H'(m||m') = H'(h'||m')$, where m' is an arbitrary bitstring. This attack was countered by encoding the length of the message as the final block in the iterated hash function. Thus, since the lengths of the two messages vary, the result of the hash function will vary in the final iteration of the hash function.

We can extend the hash function H_2 to use this idea. Recall that that hash function begins with some matrix C. We can encode the length of a message by starting with the matrix C raised to the length of the message power. Note that if one chooses $C = AB$, then the order of C will be far greater than the length of any reasonable message, and thus messages of different length will start with a different initial matrix. Thus we can define hash function $H_3 : \{0,1\}^* \to SL_2(\mathbb{F}_{2^n})$ by

$$H_3(b_0 b_1 \dots b_k) = C^k \backslash \pi(b_0) \backslash \pi(b_1) \backslash \cdots \backslash \pi(b_k).$$

The hash function H_3 is resistant to attacks which merely insert a message which hashes to the identity, as the resulting message differs in length from the original message.

Collisions in this hash function can still be thought as two distinct paths through the Cayley graph. In this case, however, the paths generally have different starting vertices. In this sense, the starting vertex is the matrix C^k, where k is the length of the message being hashed.

This does have an effect on the security properties of this hash function. Recall that collisions using the hash function H_2 can be thought of as two paths through the Cayley graph associated with the elements of $\langle A, B\rangle$ with left division as the operation, with equal starting and ending vertices. Collisions from local modification of text will need to involve bitstrings of length no less than the girth of the graph. However since in H_3 colliding messages generally have different starting vertices, the girth only provides protection from local modifications when the colliding messages have equal length.

CHAPTER 6. FUTURE WORK AND CONCLUSIONS

6.1 Future Work

Chapter 4.3.2 discussed an attack developed in [38] that uses the decomposition of the irreducible polynomial defining the field $\mathbb{F}_{2^n}$ to find group elements which have small order. However, we showed that very few irreducible polynomials have a non-trivial decomposition, thus greatly limiting the practicality of this attack.

It may be possible to extend the attack in [38] to work even when the original defining irreducible polynomial is indecomposable. It seems to be the case that for most composite integers n, there exists an irreducible polynomial of degree n over $\mathbb{F}_2$ that has a non-trivial polynomial decomposition. Suppose $\mathbb{F}_{2^n}$ is represented as $\mathbb{F}_2[x]/\langle p(x)\rangle$, and suppose we can find a second irreducible polynomial $p'(x)$ of equal degree which is decomposable. We can use the attack in [38] to find a message which hashes to a matrix with small order when the computations are performed over $\mathbb{F}_2[x]/\langle p'(x)\rangle$. An interesting question is whether we can use that result to attack the hash function when computations are performed over $\mathbb{F}_2[x]/\langle p(x)\rangle$.

There are many open questions related to variations of the Tillich-Zémor hashing scheme using quasigroups. The variations discussed in Chapter 5 create quasigroups by using non-associative binary operations in place of matrix multiplication. We could use this approach with any large quasigroup whose operation can be computed efficiency on a computer. Meyer addressed the problem of creating and representing large non-associative quasigroups in [30]. Those ideas could be used to create other quasigroup-based variations of the hashing scheme.

Also, the hash functions in Chapter 5 have the common problem that they offer little or no proof of security. These functions, along with any other variation based on quasigroups, do not have the same degree of structure as the original hash functions. Still, it may be possible

to develop proofs of security for hash functions of this design.

6.2 Conclusions

Hash functions play a vital role in maintaining the integrity of information in the digital age. As discussed in Chapter 2, hash functions are one-way, collision resistant functions that are generally built from block ciphers. Recent attacks have found weaknesses in the most commonly used hash functions today, causing some cryptographers to call for the creation of new hash functions built on mathematical principles.

Chapters 3 and 4 discussed two related hash functions based on matrix multiplication over $SL_2(\mathbb{F}_p)$ and $SL_2(\mathbb{F}_{2^n})$. In the case of the hash function over $SL_2(\mathbb{F}_p)$, the scheme quickly fell to an attack which can find factorization of the identity matrix very quickly. While this attack only directly works for a particular pair of generators for $SL_2(\mathbb{F}_p)$, alternative generating sets are unlikely to offer a significantly improved level of security.

In Chapter 4 we discussed a hash function operating over $SL_2(\mathbb{F}_{2^n})$. As field operations in $\mathbb{F}_{2^n}$ can be performed efficiently on computers, this function appears to be practical from a computational standpoint. However, as discussed in Chapter 4.3, there have been two attacks developed against this hash function. The first of these attacks creates a system of non-linear equations and uses discrete logarithms to find a collision. However, the colliding messages are extremely long, and the likelihood of finding collisions with messages of reasonable length appears to be insignificant. The second attack requires that the irreducible polynomial defining the field $\mathbb{F}_{2^n}$ be decomposable. Such polynomials can easily be avoided when choosing a representation of the field for the construction of the hash function.

Neither attack poses a credible threat to the security of the hash function. However, they are two examples of undesirable properties of this hash function and they suggest that more serious attacks could be developed. For instance, we do know that $SL_2(\mathbb{F}_{2^n})$ contains elements of small order. The fact that the attack described in Chapter 4.3.2 is capable of finding such elements quickly in limited circumstances suggests that other attacks could be developed that work in more general cases.

It should be noted that both the hash function over $SL_2(\mathbb{F}_p)$ and the hash function over $SL_2(\mathbb{F}_{2^n})$ have a security claim. The hash functions are related to the Cayley graph of the group with respect to the generators $\{A, B\}$. Specifically, colliding messages can be thought of as two paths through this graph that begin at the identity matrix and end at an arbitrary group element. At a minimum, two colliding messages must differ in at least two bits k locations apart, where k is the girth of the Cayley graph. In practice, however, this provides no real level of security.

Most attacks against these hash functions make use of the associativity of matrix multiplication. Because of this, once a message is found that hashes to the identity matrix, that message can be inserted into any other message to obtain a collision. In Chapter 5 we considered different variations of the hash function design that use quasigroups instead of the special linear group of 2×2 matrices over a finite field. These variations offer protection against known attacks, but have no proven security claims involving a minimum bound on the girth of the Cayley graphs. Despite the fact that such a claim would offer little practical security, this is troubling as hash functions based on these quasigroups appear to have no advantage over traditional hash functions, which also tend to lack practical security proofs.

None of the hash functions presented in this paper would be suitable replacements for SHA-1, MD-5 or other commonly used hash functions. All evidence suggests that these functions are slower and less secure than hash functions constructed in more traditional ways. However, hash functions constructed using algebraic structures have the potential for offering provable security claims and will continued to be studied.

APPENDIX A. Discrete Log Attack Implementation in Axiom

```
++Set up field of order 2^60

GF2:=PrimeField 2 m:=monomial(1,1)$SUP(GF2)

deg:=60;

p1:=createIrreduciblePoly(deg)$FFPOLY(GF2)

F1:=FFP(GF2,p1)

x:=index(2)$F1


++Set up deg 2 extension (order 2^(2*60))

p2:=createIrreduciblePoly(2*deg)$FFPOLY(GF2)

F2:=FFP(GF2,p2);

y:=index(2)$F2;

xhat:=x::F2

k:=(2^(deg*2)-1)/(2^deg-1);


++ Set up generators for SL_2

A:=matrix[[x,1],[1,0]];

B:=matrix[[x,x+1],[1,1]];

C:=A^160*B^258

Itwo:=matrix[[x^0,0*x],[0*x,x^0]]


++Create system of equations for a possible collision

ls:List Symbol:=[lambda1,mu1,lambda2,mu2,lambda3,mu3];
```

```
ls2:List Symbol:=[lambda1,mu1,lambda2,mu2,lambda3,mu3,new()$Symbol]

RHS:=(lambda1*A + mu1*Itwo)*(lambda2*B + mu2*Itwo)*(lambda3*A +mu3*Itwo);

p1:=RHS(1,1)-C(1,1);

p2:=RHS(1,2)-C(1,2);

p3:=RHS(2,1)-C(2,1);

p4:=RHS(2,2)-C(2,2);

p5:=determinant lambda1*A+mu1*Itwo)-1;

p6:=determinant (lambda2*B+mu2*Itwo)-1;

p7:=determinant(lambda3*A+mu3*Itwo)-1;

polys:=[p1,p2,p3,p4,p5,p6,p7];


++(Attempt to) solve the system and copy a solution

soln:=solve([p1=0,p2=0,p3=0,p4=0,p5=0,p6=0,p7=0],ls);

solnum:=1;

l1:= eval(lambda1,soln(solnum))::F1;

l2:= eval(lambda2,soln(solnum))::F1;

l3:= eval(lambda3,soln(solnum))::F1;

m1:= eval(mu1,soln(solnum))::F1;

m2:= eval(mu2,soln(solnum))::F1;

m3:= eval(mu3,soln(solnum))::F1;


++let beta be a root of min poly for A

temp:=solve(t^2+(xhat)*t+1);

beta:=eval(t,temp(1))::F2;


++Check conditions on the order of A

++  These values below must be 1 for

++  the attack to work
```

```
(l1::F2*beta+m1::F2)^(order(beta))
(l3::F2*beta+m3::F2)^(order(beta))

++If so, calculate the discrete logarithms
i1:=discreteLog(beta,(l1::F2)*beta+(m1::F2))
i3:=discreteLog(beta,(l3::F2)*beta+(m3::F2))

++Now, let gamma be a root of min poly for B
temp:=solve(t^2+(xhat+1)*t+1=0);
gamma:=eval(t,temp(1))::F2;

++Check condition on the order of B
++  This value must be 1, or attack fails
(l2::F2*gamma+m2::F2)^(order(gamma))

++If so, calculate discrete logarithm
i2:=discreteLog(gamma,(l2::F2)*gamma+(m2::F2))

(A^i1*B^i2*A^i3=C)@Boolean
```

APPENDIX B. Small Order Attack Implementation in Axiom

```
++Function: revbitstring
++ Input: A positive integer, n
++ Output: the binary representation of n
++         in a list with the least significant
++         bit first
revbitstring(0)==[0]::List NonNegativeInteger;
revbitstring(1)==[1]::List NonNegativeInteger;
revbitstring(n)==cons(rem(n,2),revbitstring(shift(n,-1)));

++Function: mybitstring
++ Input: A positive integer, n
++ Output: Binary representation with the most
++         significant bit first
mybitstring(n)==reverse(revbitstring(n));

++Function: TZHash
++ Input: A list containing 0s and 1s
++ Output: The Tillich-Zemor hash value of the message
++         over the field F_{2^100}
TZHash(msg)==
    GF2:=PrimeField 2;
```

```
    m:=monomial(1,1)$SUP(GF2);
    p1:=m^100+m^75+1;
    F1:=FFP(GF2,p1);
    x:=index(2)$F1;
    A:=matrix[[x,1],[1,0]];
    B:=matrix[[x,x+1],[1,1]];
    Itwo:=matrix[[x^0,0*x],[0*x,x^0]];
    h:=Itwo;
    for i in msg repeat
        if i=0 then h:=h*A else h:=h*B
    h


++Function: mytrace
++ Input: A 2x2 matrix
++ Output: The sum of the diagnonal entries
mytrace(temp)==temp(1,1)+temp(2,2);


++Function: searchBitStrings
++ Inputs:
++     poly: a polynomial over F_2
++     degp: the degree of poly
++ Output: a bitstring whose hash value has a trace equal to poly
searchBitStrings(poly,degp)==
    local temphash;
    for j in (2^(degp-1))::Integer..(2^degp)::Integer repeat
        print(j);
        temphash:=TZHash(mybitstring(j));
        if (mytrace(temphash)=poly)@Boolean then return mybitstring(j)
```

```
    "error"


++ Function: determineOrder
++ Inputs:
++    mat: a matrix in SL_2(F_{2^100})
++    num: an upper bound on the order of mat
++ Output: the order of mat
++ Note: This function is quite slow.  There are several
++       ways to improve this function, but none are necessary
++       to demonstrate this attack
determineOrder(mat,num)==
    GF2:=PrimeField 2;
    m:=monomial(1,1)$SUP(GF2);
    p1:=m^100+m^75+1;
    F1:=FFP(GF2,p1);
    x:=index(2)$F1;
    Itwo:=matrix[[x^0,0*x],[0*x,x^0]];
    for z in 1..num::Integer repeat
        print(z);
        t:=mat^z;
        if (t=Itwo)@Boolean then return z
    "unknown"
```

BIBLIOGRAPHY

[1] Kanat S. Abdukhalikov and Chul Kim. On the security of the hashing scheme based on sl2. In *FSE '98: Proceedings of the 5th International Workshop on Fast Software Encryption*, pages 93–102, London, UK, 1998. Springer-Verlag.

[2] David R. Barton and Richard E. Zippel. A polynomial decomposition algorithm. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 356–358, New York, NY, USA, 1976. ACM Press.

[3] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: the case of hashing and signing. In Yvo G. Desmedt, editor, *CRYPTO '94*, pages 216–233. Springer, 1994. Lecture Notes in Computer Science No. 839.

[4] E. R. Berlekamp. Factoring polynomials over large finite fields. In *SYMSAC '71: Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, page 223, New York, NY, USA, 1971. ACM Press.

[5] B. Buchberger. A theoretical basis for the reduction of polynomials to canonical forms. *SIGSAM Bull.*, 10(3):19–29, 1976.

[6] Donghoon Chang. Preimage attack on hashing with polynomials proposed at icisc'06. Cryptology ePrint Archive, Report 2006/411, 2006. `http://eprint.iacr.org/`.

[7] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In J. Feigenbaum, editor, *CRYPTO '91*, pages 470–484. Springer, 1992. Lecture Notes in Computer Science No. 576.

[8] Don Coppersmith. Evaluating logarithms in $gf(2^n)$. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 201–207, New York, NY, USA, 1984. ACM Press.

[9] Don Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–593, 1984.

[10] Ivan Damgård. Collision free hash functions and public key signature schemes. In *EUROCRYPT*, pages 203–216, 1987.

[11] Ivan Bjerre Damgård. A design principle for hash functions. In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 416–427, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[12] Richard Drews Dean. *Formal aspects of mobile code security*. PhD thesis, 1999. Adviser-Andrew Appel.

[13] Jintai Ding, Jason E. Gower, and Dieter S. Schmidt. Zhuang-zi: A new algorithm for solving multivariate polynomial equations over a finite field. Cryptology ePrint Archive, Report 2006/038, 2006. `http://eprint.iacr.org/`.

[14] D. S. Dummit and R. M. Foote. *Abstract Algebra*. John Wiley and Sons, third edition, 2004.

[15] J. C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional groöbner bases by change of ordering. *J. Symb. Comput.*, 16(4):329–344, 1993.

[16] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., New York, NY, USA, 2003.

[17] Willi Geiselmann. A note on the hash function of tillich and zémor. In *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 51–52, London, UK, 1996. Springer-Verlag.

[18] J. K. Gibson. Discrete logarithm hash function that is collision free and one way. *IEE Proceedings-E*, pages 407–410, 1991. No. 138.

[19] Jaime Gutierrez, Tomás Recio, and C. Ruiz de Velasco. Polynomial decomposition algorithm of almost quadratic complexity. In *AAECC-6: Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 471–475, London, UK, 1989. Springer-Verlag.

[20] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the wang et al. md5 collision. Cryptology ePrint Archive, Report 2004/264, 2004. `http://eprint.iacr.org/`.

[21] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

[22] Antoine Joux and Reynald Lercier. The function field sieve is quite special. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 431–445, London, UK, 2002. Springer-Verlag.

[23] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.

[24] Dexter Kozen and Susan Landau. Polymonial decomposition algorithms. *J. Symb. Comput.*, 7(5):445–456, 1989.

[25] Niels Lauritzen. *Concrete Abstract Algebra*. Cambridge University Press, Cambridge, 2003.

[26] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997.

[27] Stefan Lucks. A failure-friendly design principle for hash functions. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.

[28] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[29] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.

[30] Kristen Ann Meyer. *A new message authentication code based on the non-associativity of quasigroups*. PhD thesis, 2006. Adviser-Clifford Bergman.

[31] NIST. *FIPS PUB 180-1: Secure Hash Standard*. Gaithersburg, MD, USA, 1995.

[32] NIST. *FIPS PUB 197: The Advanced Encryption Standard*. Gaithersburg, MD, USA, 2001.

[33] Bill Page. Axiom. www.axiom-developer.org. accessed March 28, 2007.

[34] Bart Preneel. The state of cryptographic hash functions. In Ivan Damgård, editor, *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 158–182. Springer, 1998.

[35] R. Rivest. The md5 message-digest algorithm, 1992.

[36] V. Shpilrain. Hashing with polynomials. In *ICISC 2006*, pages 22–28. Springer, 2006. Lecture Notes in Computer Science No. 4296.

[37] Joseph H. Silverman. Fast multiplication in finite fields gf(2^n). In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 122–134, London, UK, 1999. Springer-Verlag.

[38] Rainer Steinwandt, Markus Grassl, Willi Geiselmann, and Thomas Beth. Weaknesses in the $sl_2(\mathbb{F}_{2^n})$ hashing scheme. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 287–299, London, UK, 2000. Springer-Verlag.

[39] Marc Stevens, Arjen Lenstra, and Benne de Weger. Target collisions for md5 and colliding x.509 certificates for different identities. Cryptology ePrint Archive, Report 2006/360, 2006. `http://eprint.iacr.org/`.

[40] Douglas Stinson. *Cryptography: Theory and Practice,Second Edition.* CRC/C&H, 2002.

[41] Emmanuel Thomé. Computation of discrete logarithms in $\mathbb{F}_2^{607}$. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 107–124, London, UK, 2001. Springer-Verlag.

[42] Jean-Pierre Tillich and Gilles Zémor. Group-theoretic hash functions. *Lecture Notes In Computer Science*, 781:90–110, 1993.

[43] Jean-Pierre Tillich and Gilles Zémor. Hashing with SL_2. *Lecture Notes in Computer Science*, 839:40–49, 1994.

[44] J. von zur Gathen. Functional decomposition of polynomials: the tame case. *J. Symb. Comput.*, 9(3):281–299, 1990.

[45] J. von zur Gathen. Functional decomposition of polynomials: the wild case. *J. Symb. Comput.*, 10(5):437–452, 1990.

[46] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Report 2004/199, 2004. `http://eprint.iacr.org/`.

[47] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[48] Gilles Zémor. Hash functions and cayley graphs. *Designs, Codes and Cryptography*, 4(3):381–394, 1994.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to those who helped me as I wrote this thesis.

First I would like to thank Dr. Cliff Bergman for her guidance and support through the research and the writing of this thesis. Next, I would like to thank my committee members, Drs. Sung Song and Siu-Hung Ng, for their assistance with this thesis and my coursework. Thanks also to Dr. Jonathon D.H. Smith for his helpful insights on this work.

Finally, I would like to thank my friends and family for their kind words, support and encouragement throughout this work. You were always there for me when I needed someone to lean on.